CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB
FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS - FATECS
CURSO DE ENGENHARIA DE COMPUTAÇÃO
PROJETO FINAL

VINÍCIUS RODRIGUES TONHÁ

SISTEMA DE AUTENTICAÇÃO DE COMPRAS ELETRÔNICAS UTILIZANDO TECNOLOGIA *SMART CARD* E BIOMETRIA DIGITAL

Orientador: Ms. C. Prof. Francisco Javier de Obaldía Díaz

Brasília - DF,
Junho de 2011

VINÍCIUS RODRIGUES TONHÁ
RA: 2061532/7

SISTEMA DE AUTENTICAÇÃO DE COMPRAS ELETRÔNICAS UTILIZANDO TECNOLOGIA *SMART CARD* E BIOMETRIA DIGITAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Ms. C. Prof. Francisco Javier de Obaldía Díaz.

Brasília-DF,
Junho de 2011

VINÍCIUS RODRIGUES TONHÁ
RA: 2061532/7

SISTEMA DE AUTENTICAÇÃO DE COMPRAS ELETRÔNICAS UTILIZANDO TECNOLOGIA *SMART CARD* E BIOMETRIA DIGITAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Ms. C. Prof. Francisco Javier de Obaldía Díaz.

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas-FATECS.

_____

Prof. Abiezer Amarilia Fernandez
Coordenador do Curso

Banca Examinadora:

_____

Prof. Francisco Javier de Obaldía Díaz, Mestre em Engenharia Elétrica
Orientador - UniCEUB

_____

Prof. Maria Marony Souza Farias Nascimento, Mestre em Engenharia Elétrica
UniCEUB

_____

Prof. Luis Cláudio Lopes de Araújo, Mestre em Matemática Pura
UniCEUB

DEDICATÓRIA

Dedico este trabalho de conclusão de curso, bem como todos os passos que me trouxeram até aqui às pessoas mais importantes da minha vida: minha mãe, Geane Rodrigues da Silva Tonhá, ao meu pai, Manoel Tonhá de Oliveira e aos meus irmãos, Matheus e Amanda.

"Feliz o homem que pôs
sua esperança no Senhor"
Bíblia Sagrada - Salmos 39, 5

AGRADECIMENTOS

Em primeiro lugar agradeço a minha mãe pelo apoio incondicional e por ter me proporcionado todas as oportunidades apesar da dificuldade. Não só esta, mas todas as demais conquistas em minha vida serão dedicadas a ela.

Agradeço ao meu pai pelo exemplo de homem e por ter me ensinado os valores que me trouxeram até aqui.

Agradeço a minha namorada, Naira, pelo apoio e compreensão nos momentos de dificuldade e pelo incentivo de sempre.

Agradeço a todos os professores do curso de Engenharia da Computação, não só pelos ensinamentos acadêmicos, mas por terem contribuído para minha formação como homem e cidadão.

Por fim, agradeço a todos os colegas de classe que fizeram parte desta jornada, em especial aos amigos Guilherme, Camilla, Vanessa, Leonardo Lima, Marco Aurélio, Gabriel, Reinaldo, Victor, Diogo, Paulo Gabriel e Thiago.

LISTA DE FIGURAS

LISTA DE QUADROS E TABELAS

LISTA DE ABREVIATURAS

CNPJ - Cadastro Nacional de Pessoas Jurídicas

CPF - Cadastro de Pessoas Físicas

DNA - *Deoxyribonucleic acid,* em português, Ácido desoxinorribonucleico

POS - *Point of Sale*, em português, Ponto de venda

PDV - Ponto de venda

ID - Impressão Digital

CF - Crista Final

CB - Crista Bifurcada

SDK - S*oftware Development Kit,* em português, Kit de Desenvolvimento de *Software*

FBI - *Federal Bureau of Investigation*, em português, Escritório Federal de Investigação

AFIS - *Automated Fingerprint Identification System*, em português, Sistema de Identificação Automatizada da Impressão Digital.

ATM - *Automatic Teller Machine*

ISO - *International Organization for Standardization*, em português, Organização Internacional para Padronização.

ROM - *Read Only Memory*, em português, Memória Somente de Leitura.

EEPROM - *Electrically-Erasable Programmable Read-Only Memory*.

RAM - *Random Access Memory*, em português, Memória de Acesso Randômico.

APDU - *Application Protocol Data Units.*

TPDU - *Transmission Protocol Data Units.*

ATR - *Answer to Reset.*

JCVM - *Java Card Virtual Machine.*

JCRE - *Java Card Runtime Environment.*

API - *Application Programming Interface.*

JVM - *Java Virtual Machine.*

JCDK - *Java Card Development Kit.*

PIN - *Personal Identification Number*

RESUMO

Este trabalho tem por objetivo a integração de duas tecnologias bastante difundidas nos dias de hoje: *smart card* e biometria digital. Objetiva acima de tudo propor uma nova metodologia para mitigar problemas de segurança relacionados a transações de compra nas funções débito e crédito em estabelecimentos comerciais. Através dessas duas tecnologias, procura-se minimizar problemas e transtornos relacionados a fraudes diversas, roubos e clonagens de cartões, captura de senhas, utilização de cartões por terceiros, entre outros.

A solução foi desenvolvida utilizando os seguintes equipamentos: um computador, um leitor biométrico, uma leitora e gravadora de *smart cards* e um cartão *smart card* com suporte a tecnologia *Java Card 2.2.1*. O sistema divide-se em duas partes: cadastramento de usuários, capturando sua impressão digital e seus dados cadastrais e sua posterior vinculação a um cartão inteligente através da inserção de seus dados no *chip* do cartão; e a autenticação da compra realizada pelo cliente em questão, através da simulação de um terminal POS e verificação da impressão digital através do *smart card*. A compra é efetivada se, na comparação 1:1, o novo exemplar recolhido pela leitora instalada no terminal POS simulado for compatível com a digital previamente armazenada em banco de dados.

O sistema é baseado em duas metodologias de acesso: aquilo que se possui (o cartão *smart card*) e aquilo que se é (característica biométrica), contemplando duas tecnologias já consagradas no mercado.

Palavras Chave: Biometria, impressão digital, *smart cards, Java Card*, sistema financeiro, compras eletrônicas.

ABSTRACT

This project aims the integration of two technologies widely spread today: smart card and biometric identification of fingerprints. The scope is furthermore to present a new methodology for solving security issues related to purchase transactions in debit and credit functions in commercial establishments. Through these two technologies, problems and inconveniences related to fraud, theft and cloning of cards, capture of passwords, use of cards by third parties, among others are sought to be solved.

The solution was developed using the following equipment: a computer, a biometric reader, a reader and burner of smart card and a smart card supporting technology of Java Card 2.2.1. The system is divided into two parts: registration of users, capture of their fingerprints and registration data, afterwards its linking to a smart card by recording their data in a chip card; and the authentication of the purchase performed by the customer in question, through a simulation of a POS terminal and the verification of the fingerprint through the smart card. The purchase is confirmed, if after the confrontation 1:1, the client's fingerprint already registered is compatible with the specimen collected by the reader at the installed simulated POS terminal.

The system is based on two methods of access: what is in possession (the smart card) and what it is (biometric characteristic), comprising two technologies already established in the market.

Keywords: Biometrics, fingerprint, smart cards, Java Card, financial system, electronic purchases.

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

Segundo INÁCIO, podemos definir fraude da seguinte maneira:

> "Ato ou efeito de fraudar, de modificar ou alterar um produto ou esconder a qualidade viciada deste, com objetivo de lucro ilícito; burla, dolo; engano, etc. Outros autores definem como "qualquer ato ardiloso, enganoso de má fé, com intuito de lesar ou ludibriar outrem, ou de não cumprir determinado dever; logro, falsificação de marcas ou produtos industriais, de documentos etc. Introdução clandestina de mercadorias estrangeiras sem o pagamento dos devidos tributos à alfândega; iludir, falta, crime, delito" (INÁCIO, 2011).

A utilização de meios eletrônicos de pagamento já está consolidada no Brasil e no mundo. O uso de cartões para realização de compras em detrimento do uso de dinheiro, cheque, boleto ou outras modalidades convencionais de pagamento é uma forte tendência que vem sendo amplamente incentivado pelas instituições financeiras e pelo comércio.

Para garantir a segurança e confiabilidade dessas transações, várias medidas foram adotadas e vem sendo aperfeiçoadas ao longo do tempo tornando-as mais rigorosas e eficazes. Todavia, os fraudadores não tem permanecido atrás, sempre inventando novas modalidades de fraude ou *modus operandi*, principalmente os baseados nas falhas de utilização dos clientes.

Dessa forma, hoje as fraudes com cartões são um fenômeno comum, normalmente envolvendo pequenas quantias. Porém, em alguns casos podem se tornar graves problemas tanto paras os seus proprietários quanto para as instituições financeiras emitentes destes cartões.

O objetivo deste trabalho é o estudo da integração de duas soluções de controle de acesso amplamente utilizadas nos dias de hoje: s*mart cards* e autenticação biométrica da impressão digital, para que, conjuntamente, ofereçam uma nova forma de acesso as compras no crédito e débito, tornando-as mais seguras e eficazes. Além disso, contempla o

desenvolvimento de aplicação integrando essas tecnologias. O baixo índice de falha das biometrias e a segurança contra clonagem obtida com os cartões inteligentes é o diferencial para a criação dessa solução de autenticação de compras eletrônicas.

1.1 - Motivação

O grande aumento do uso de meios eletrônicos para realização de compras trouxe consigo uma maior visibilidade e uma maior ação de pessoas má intencionadas. Essas pessoas visam a burla dos sistemas financeiros e a obtenção de lucros com tais práticas.

As diversas fraudes sofridas contra os sistemas financeiros e seus usuários mostram a necessidade de uma metodologia de autenticação de compras mais seguro, o que pode ser possibilitado através da junção de duas tecnologias de controle de acesso que vêm demonstrando sua eficácia ao longo do tempo.

Através da biometria digital e dos cartões *smart card*, procura-se minimizar problemas e transtornos relacionados a fraudes diversas, roubos e clonagens de cartões, captura de senhas, utilização de cartões por terceiros, entre outros.

1.2 - Objetivos

Apresentar uma nova forma de acesso e autenticação das compras eletrônicas realizadas nas funções crédito e débito baseado na junção das tecnologias *smart card* e biometria digital.

1.2.1 - Objetivos Específicos

- Desenvolvimento do *software* para leitura, cadastramento e armazenamento da impressão digital e das informações cadastrais do cliente;

- Desenvolvimento do *software* para vinculação do usuário a um cartão, através da gravação dos seus dados no *chip* do cartão;

- Simulação do ambiente de compras e de seus diversos fluxos;

- Integração da leitora/gravadora de cartões e leitora de impressões digitais com os *softwares* desenvolvidos.

1.3 - Estrutura da Monografia

Este trabalho de conclusão de curso está disposto da seguinte maneira:

Capítulo 1: Introdução. Como aqui disposto, são apresentados os objetivos gerais e específicos, motivação e como será exposto o mesmo.

Capítulo 2: Apresentação do problema. São apresentados dados sobre o mercado financeiro nacional, migração para meios eletrônicos de pagamento, fraudes aplicadas contra o sistema financeiro nacional, metodologias para prevenção de fraudes e controle de acesso e os limites das soluções atuais.

Capítulo 3: Referencial Teórico. Aqui são abordados os conceitos básicos que fundamentam o projeto. Definições sobre função crédito e débito e seu funcionamento. Conceitos teóricos sobre impressão digital humana, biometria e seus tipos, cartões *smart card*, seu uso e aplicação.

Capítulo 4: Desenvolvimento do Sistema de Autenticação. Aqui são apresentados os dados referentes ao projeto em si, *software* e *hardware*. O procedimento, metodologia, tecnologias e equipamentos utilizados são detalhados neste capítulo. Aplicabilidade e estimativa de custos também se encontram neste capítulo.

Capítulo 5: Aplicação do Sistema e Resultado dos Testes. Apresentação e análise dos testes realizados, visando comprovar o funcionamento e efetividade do sistema.

Capítulo 6: Conclusão. Conclusão do projeto, comentários e análises finais e sugestões para trabalhos futuros.

CAPÍTULO 2 – APRESENTAÇÃO DO PROBLEMA

2.1 - Mercado e Sistema Bancário em Números

O uso dos cartões para realização de compras tanto na função crédito como na função débito cresce a cada dia. Os serviços que foram criados visando maior praticidade, rapidez e autonomia dos clientes, hoje são utilizados diariamente por milhões de pessoas ao redor do mundo.

No quadro 2.1 é apresentado a evolução nos números relacionados ao uso dos cartões de 2000 a 2009. De acordo com a tabela, no ano de 2009, 2.5 bilhões de transações de compras na função crédito foram realizadas. Essas transações envolveram o montante de 256 bilhões de reais, um crescimento de aproximadamente 19% com relação ao ano de 2008 e de 469% com relação ao ano de 2000, quando os dados começaram a ser computados.

Segundo FEBRABAN (2010, p.8), a quantidade de cartões em circulação no mercado pulou de 29 milhões para 136 milhões entre os anos de 2000 e 2009, um crescimento de 369%.

QUADRO 2.1 – TRANSAÇÕES COM CARTÕES DE CRÉDITO

Período	Unidade	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	Variação 2009/2008
Cartões de crédito	*milhões*	29	38	42	45	53	68	82	104	124	136	*10%*
Transações com cartões de crédito	*bilhões*	0,6	0,7	0,8	0,9	1,1	1,3	1,6	1,9	2,2	2,5	*14%*
Valor total de transações com cartões	*R$ bilhões*	45	60	69	83	95	115	142	174	215	256	*19%*

FEBRABAN

FONTE: FEBRABAN, 2010, P. 8

Seguindo essa tendência, o valor médio das transações efetuadas no sistema financeiro nacional com cartões de crédito cresceu 30% entre os anos de 2000 e 2008, superando a marca de R$ 100,00 por transação em 2009 (FEBRABAN, 2010, p.8).

Da mesma forma, segundo ABECS (2010), até setembro de 2010 o faturamento com cartões na função débito ultrapassou o montante de R$ 157 bilhões, com um total de quase 2,84 bilhões de transações.

Os números apresentados na tabela 2.1 demonstram que o uso desses serviços apresenta forte aumento nos últimos anos e que se trata de uma tendência irreversível, que irá se expandir ainda mais tendo em vista as diversas vantagens que apresenta para todos os agentes econômicos.

2.2 - Migração para meios eletrônicos

Hoje, as instituições financeiras, redes e administradoras de cartões e os próprios estabelecimentos comerciais vêm estimulando o uso de compras eletrônicas em detrimento do uso de dinheiro vivo, cheque, boleto e outros meios convencionais de pagamento.

Esse tipo de incentivo vem trazendo resultados para os bancos, que tiveram seus custos operacionais e logísticos reduzidos, o que levou também a uma maior autonomia dos clientes, uma vez que podem se "auto-atender", contribuindo também para o "desafogamento" das agências bancárias. Para o comércio e economia representa uma verdadeira revolução, dada a enorme expansão do crédito que possibilita, no caso dos cartões de crédito, e o incentivo à circulação da moeda e impulsionamento do comércio e do desenvolvimento econômico (ANDRADE, 2002).

A figura 2.1 ilustra o comportamento das transações bancárias por origem até o ano de 2008, demonstrando a crescente tendência a utilização de canais eletrônicos e do auto-atendimento.

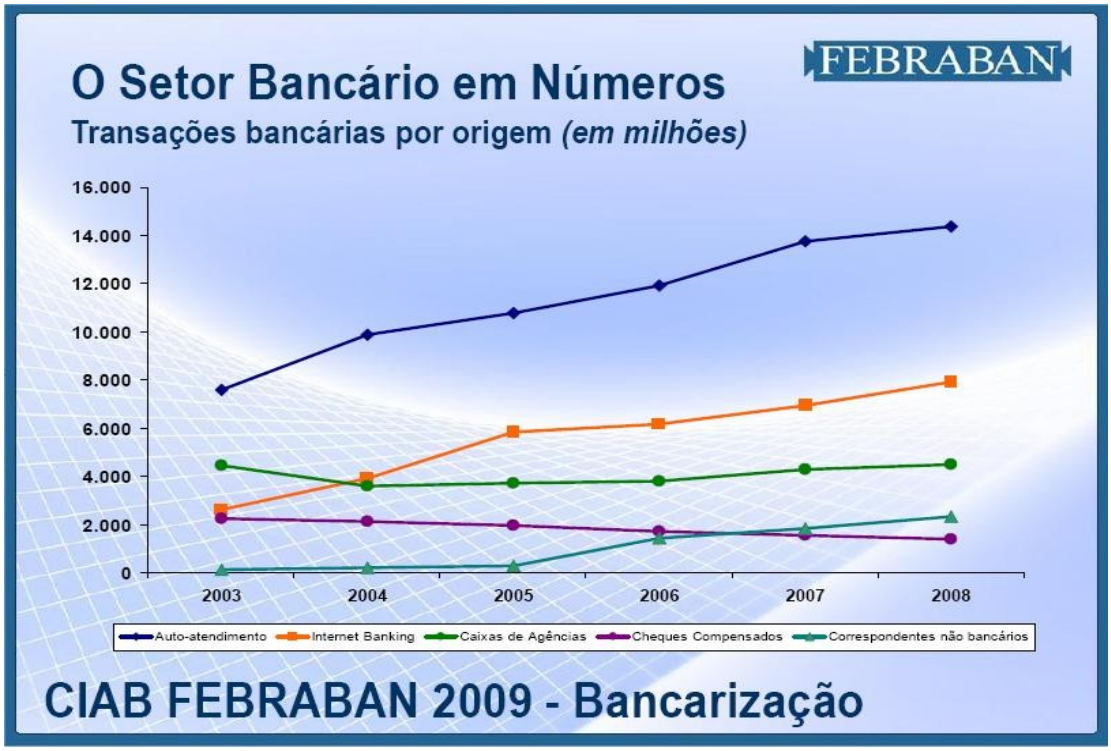

FIGURA 2.1 – TRANSAÇÕES POR ORIGEM / FONTE: FEBRABAN, 2009, P.9

2.3 - Segurança das transações

2.3.1 - Métodos de fraude aplicados com cartões

O grande aumento do uso de meios eletrônicos para movimentações financeiras trouxe consigo uma maior visibilidade e uma maior ação de pessoas má intencionadas que visam a burla desses sistemas e a obtenção de lucros com tais práticas. Segundo FRAUDES (2009), as fraudes contra cartões provêm de diversas fontes e podem ser aplicadas basicamente das seguintes formas:

- Golpe da Troca de Cartões: substituição dolosa do cartão magnético com quebra involuntária do sigilo da senha. Nesta modalidade de golpe, o cliente ao utilizar os terminais de Auto Atendimento ou POS, é interceptado por pessoa, normalmente bem vestida, falante e cortês que, na intenção de ajudá-lo no uso do equipamento, observa a

digitação de sua senha e, rapidamente troca o cartão em seu poder sem que este perceba de imediato. A partir daí, realizam transferências, compras e sacam valores existentes em conta corrente, poupança, cartão e aplicações;

- Golpe da Violação da Senha por telefone: o cliente recebe ligação telefônica de pessoa que se identifica como funcionário do Banco. Utilizando argumentos variados (devolução de impostos ou restituição de Imposto de Renda, oferta de brindes, premiações, etc.), solicita que o cliente digite sua senha no teclado do telefone;

- Cartões Falsificados ou Clonados: cartões de emissão fraudulenta cujos dados são copiados de cartões existentes, de carbonos ou comprovantes de venda, etc. Existe também o "*skimming"* ou cópia de tarjas magnéticas de cartões válidos ("clone"), cujos dados do plástico são obtidos através da instalação de equipamento espúrio em terminais eletrônicos;

- A falsificação envolve o uso no comércio e também saques com uso de senha no exterior, tendo em vista que fora do ambiente do banco, não existe a exigência da digitação do código de acesso (três letras);

- Auto Fraude: o próprio cliente pede um cartão sem intenção de efetuar o pagamento, realiza gastos elevados e depois desaparece, não sendo localizado;

- Proposta Fraudulenta/Falsidade Ideológica: o fraudador se utiliza de documentos roubados ou com numeração inexistente para abrir conta e solicitar cartão. O verdadeiro dono do documento normalmente surge quando, por inadimplência, seu nome é incluído no Serasa/SPC. O termo falsidade ideológica é utilizado genericamente no mercado, englobando a auto fraude e a proposta fraudulenta.

2.3.2 - Medidas atuais de segurança

Segundo FRAUDES (2009), como medida para evitar fraudes e burlas aos sistemas eletrônicos de pagamento atualmente se utilizam três medidas básicas de segurança:

- Identificação minuciosa dos clientes: efetuar operação mediante apresentação de documento oficial com foto; validação do nome do cliente em relação ao CPF/CNPJ junto ao site da Receita Federal; Validação de dados do cliente junto a sistemas de proteção ao crédito (SPC, Serasa, etc.). A figura 2.2 mostra o documento de identidade ou R.G, considerado o documento mais completo para realização de verificações;

FIGURA 2.2 - DOCUMENTO DE IDENTIDADE/

FONTE: CONECTADOS, 2010

- Verificação dos itens de segurança do cartão: verificar o número gravado em relevo na frente do cartão, este deve ser igual ao número impresso no verso e no comprovante de vendas emitido pelo Terminal POS. Ele contém 16 dígitos; Conferir a data de validade do cartão; Verificar se o cartão possui a marca de segurança em relevo ao lado do holograma; Observar o holograma; Verificar o código de segurança que é formado pelos três últimos dígitos subsequentes ao número do cartão, localizados no painel de assinatura, no verso do cartão; Se o cartão não estiver assinado, pedir ao cliente que o

assine e comparar com a assinatura do seu RG. Nas figuras 2.3 e 2.4 pode-se ver os itens de verificação utilizados para constatar a legitimidade de um cartão;

FIGURA 2.3 – ITENS DE SEGURANÇA DO CARTÃO / FONTE: FRAUDES, 2009

FIGURA 2.4 – ITENS DE SEGURANÇA DO CARTÃO - VERSO / FONTE: FRAUDES, 2009

- Utilização da tecnologia *Smart Card*: exige que a transação seja feita mediante senha; considerado virtualmente impossível de ser clonado; podem ser autenticados sem conexão; Os cartões com c*hip* podem armazenar dados de forma segura e sigilosa, permitindo que a verificação do PIN (número de identificação pessoal ou senha do cartão) possa ser feita internamente. Na figura 2.5 pode-se ver um exemplo de cartão *smart card,* que se caracteriza pela presença do *chip* embutido no lado esquerdo do plástico.

FIGURA 2.5 – CARTÃO COM CHIP – SMART CARD / FONTE: FRAUDES, 2009

2.3.3 - Limites de resolução dos métodos atuais

Mesmo com toda a tecnologia hoje envolvida, o investimento pesado por parte das instituições financeiras e a ação das entidades governamentais por meio de leis que regulamentam as atividades do setor, não se pode garantir 100% de confiabilidade no que se refere a garantia de que apenas o próprio dono do cartão seja capaz de realizar a transação.

Um bom exemplo é a lei 4.132 de 02/05/2008, do deputado Raad Massouh, que segundo ASA (2008), obriga os comerciantes a exigirem documento de identificação com foto

nas compras realizadas com cartão nas funções débito e crédito e que vêm sendo sistematicamente descumprida.

Segundo FRAUDES (2009), com medo de perder o cliente, ou não criar constrangimentos, lamentavelmente, o comércio relaxa no processo de identificação, acentuando o problema por conta de uma cultura equivocada da qual todos fazemos parte, que é a de ficarmos “chateados” quando alguém nos identifica corretamente.

Como consequência disso, temos os prejuízos financeiros e morais sofridos pelos usuários de cartão e o prejuízo das próprias instituições financeiras que, como responsáveis pela segurança das transações, têm que arcar com os custos financeiros das fraudes aplicadas, sem contar os danos de imagem e a perda da credibilidade desse tipo de serviço (ABECS, 2010).

Assim, conclui-se não haver hoje nas transações de compras realizadas nas funções débito e crédito, nenhum meio totalmente confiável de garantir que apenas o proprietário do cartão seja capaz de realizar uma transação.

Nesse contexto, o ambiente bancário, as instituições financeiras e por sua vez as transações realizadas pelos seus clientes, necessitam de uma metodologia que garanta o acesso com segurança e confiabilidade e que torne inviável os ataques e fraudes existentes atualmente.

2.4 - Métodos de Autenticação

Segundo LEONCIO (2006), a identificação de um usuário e sua autenticação com segurança são premissas básicas para que se possa permitir o acesso de um cliente a uma determinada transação eletrônica. A identificação diz a um sistema quem você é, enquanto que a autenticação provê mecanismo eficiente de provar que você realmente é quem diz ser.

Dessa forma, necessita-se que a autenticação seja feita com base em alguma característica única, que somente o usuário identificado seja capaz de fornecer. As principais formas de validação de autenticidade são baseadas nos seguintes conceitos:

- “Algo que você sabe”;
- “Algo que você tem”;
- “Algo que você é”.

Cada um desses métodos apresenta vantagens e desvantagens, dependendo da finalidade e do nível de segurança desejados (LEONCIO, 2006).

Algo que você sabe: É o método encontrado no uso de senhas ou PINs, como para acesso à caixa postal de e-mail, senhas bancárias, senhas utilizadas no local de trabalho, etc. Apesar de ser o mais usado e também o mais antigo método de autenticação, ele contém algumas deficiências que não o tornam plenamente confiável. A principal é que ele se baseia na capacidade dos usuários de memorizar senhas, quase sempre de tamanhos e características diferentes (LEONCIO, 2006).

LEONCIO (2006) ressalta que a maioria das pessoas acaba por criar senhas com características de fácil dedução, como nomes, sobrenomes, números de documentos, placas de carros, número de telefones e datas, por falta de imaginação ou para se proteger contra o esquecimento. Outro razão é que essas senhas podem ser facilmente obtidas por pessoas mal intencionadas com o objetivo de tentar se autenticar como se fossem o titular da senha.

O baixo custo da sua implementação pode ser apontada como principal vantagem, pois pode ser usada em qualquer tipo de ambiente, sem a necessidade de *hardwares* especiais (LEONCIO, 2006).

Algo que você tem: LEONCIO (2006) cita os cartões magnéticos como o exemplo mais conhecido. Os sistemas de autenticação que se baseiam neste método, utilizam cartões ou *tokens* para validar seus usuários. Os clientes para terem acesso a compras eletrônicas, por exemplo, precisam portar um cartão juntamente com uma senha previamente cadastrada. Esse método baseia-se no fato de que apenas o titular de determinado cartão é quem vai utilizá-lo. As desvantagens desse método são:

- Os cartões podem ser facilmente perdidos ou roubados (devido ao seu pequeno tamanho);
- É caro e precisa de *hardwares* especiais para ler os cartões.

O ponto positivo desse método é que ele agrega maior nível de segurança às transações, pois considera o uso de cartão e senha ao mesmo tempo. Diante disso, uma pessoa mal intencionada agora precisa, além da senha, também do cartão para poder se passar por alguém (LEONCIO, 2006).

Algo que você é: Aqui se evidencia o uso da biometria. A biometria torna possível autenticar um indivíduo a partir de alguma característica em sua anatomia, sendo possível identificar o usuário sem a necessidade de este ter que memorizar ou guardar consigo alguma coisa, bastando apenas alguma característica já intrínseca a ele para a autenticação. A principal desvantagem é que seu uso depende da aquisição de *hardwares* especiais para capturar as informações dos indivíduos (LEONCIO, 2006).

Tendo em vista as necessidades apresentadas, propõe-se uma aplicação que além do *smart card,* utilize a metodologia baseada no que você é, devido a maior confiabilidade proporcionada pelo uso de biometrias.

CAPÍTULO 3 - REFERENCIAL TEÓRICO

Nas próximas seções serão apresentados conceitos importantes para o desenvolvimento do projeto.

3.1 - Funções Utilizadas em Compras Eletrônicas

3.1.1 - Função Crédito

De acordo com ABECS (2010), a função crédito caracteriza-se pela utilização do cartão a débito da conta cartão, ou seja, para pagamento na data do vencimento da fatura, observado o limite de crédito para compras ou saques definido para o portador, podendo ser utilizada para:

- Compras: de bens e serviços, no país e no exterior, em estabelecimentos credenciados às redes das respectivas bandeiras, por meio de terminais eletrônicos POS, maquinetas manuais, via telefone e Internet. Para utilizá-la, o portador deve optar pela função crédito no ato da compra;

- Saques: na conta cartão, por meio dos terminais de Auto-Atendimento.

Para este trabalho se focará apenas a utilização em terminais eletrônicos POS.

O cartão de crédito é uma criação relativamente recente, tendo surgido no início do século XX. A primeira ideia a se assemelhar com os atuais cartões de crédito foram os "cartões de credenciamento" (*retail cards*) emitidos por alguns hotéis europeus, a partir de 1914, para identificar seus bons clientes. Os fregueses habituais recebiam um cartão, que servia como sua identificação nas futuras hospedagens, e que garantia vantagens como deixar débitos pendentes para pagamento na próxima estada no hotel (ANDRADE, 2002).

A partir de 1920, redes de postos de gasolina nos Estados Unidos, como a Texaco e a Exxon, passaram a emitir cartões semelhantes.

Segundo ANDRADE (2002), só depois da II Guerra Mundial surgiram os primeiros cartões de crédito propriamente ditos, tais como os conhecemos hoje, emitidos por uma empresa especialmente criada para este fim. Os bens não são adquiridos junto à empresa emissora do cartão, mas em uma rede de empresas afiliadas a ela. A emissora do cartão é mera intermediária, financiando as vendas feitas junto às afiliadas.

O primeiro cartão de crédito como se conhece hoje foi o Diners Club, surgido em 1949. Inicialmente restrito a uma rede de hotéis e restaurantes afiliados. Logo o leque de opções se estendeu a diversos tipos de empresas (ANDRADE, 2002).

Em 1958, a American Express, originalmente uma agência de viagens, também criou um cartão semelhante. A partir daí, começaram a surgir várias outras empresas com a mesma finalidade (ANDRADE, 2002).

3.1.2 - Função Débito

O cartão de débito foi criado em 1951 por Franklin Bank, nos Estados Unidos. Sua ideia era utilizar o saldo em conta corrente do cliente para efetuar a compra, diferentemente do conceito até então existente para compras na função crédito (ANDRADE, 2002).

Segundo ABECS (2010), a função débito caracteriza-se pela utilização do cartão de clientes correntistas em terminais eletrônicos das respectivas redes com débito imediato em conta corrente. Pode ser utilizada para:

- Compras: de bens e serviços, no país e no exterior, em estabelecimentos credenciados às redes das respectivas bandeiras, por meio de terminais eletrônicos POS. O portador deve optar pela função débito no ato da compra;

- Saques: a débito da conta-corrente ou poupança.

Para este trabalho se focará apenas a utilização em terminais eletrônicos POS.

3.1.3 - Fluxo de uma transação

Para entender o fluxo de uma transação primeiramente tem-se que estabelecer alguns conceitos:

- Emitente: Banco ou instituição financeira que emite o cartão.

- Bandeira: administradoras de cartões. As mais conhecidas são Visa, Mastercard e American Express.

- Adquirente: empresa que fornece o serviço de rede para uma ou mais bandeiras. No Brasil os mais conhecidos são Cielo (antiga VisaNET) e a Redecard.

Segundo ABECS (2010), quando se faz uma transação com um cartão, seja ele de crédito ou de débito, utiliza-se um terminal que lê ou a tarja magnética ou o *chip* do cartão. Este terminal pode ser um POS (maquineta que normalmente utiliza-se em restaurantes, postos de gasolina, etc.) ou um PDV (por exemplo, um terminal de supermercado). A diferença entre os equipamentos é que o POS é de responsabilidade do adquirente e o PDV é um sistema fechado.

Quando se faz uma compra em um estabelecimento é necessário passar o cartão no terminal POS de um adquirente "Z" (o que tiver feito contrato com aquele estabelecimento). O terminal e o *chip* do cartão vão trocar informações, essa troca de informações inclui uma autenticação e uma validação de senha (ABECS, 2010).

Passada essa etapa, o terminal envia os dados recolhidos da transação e do cartão para a bandeira específica. Lá, sistemas especializados fazem a validação da compra e identificam se esta está dentro dos padrões, ou seja, se apresenta riscos ou desvios de comportamento, além de fazer validações relativas aos dados do cliente em si (INÁCIO, 2011).

Na figura 3.1, tem-se um cartão da bandeira “X” emitido pelo banco “Y” e um adquirente “Z”. A empresa “Z” já tem configurado em todo o seu sistema o caminho que uma transação da bandeira “X” deve fazer para chegar ao seu destino.

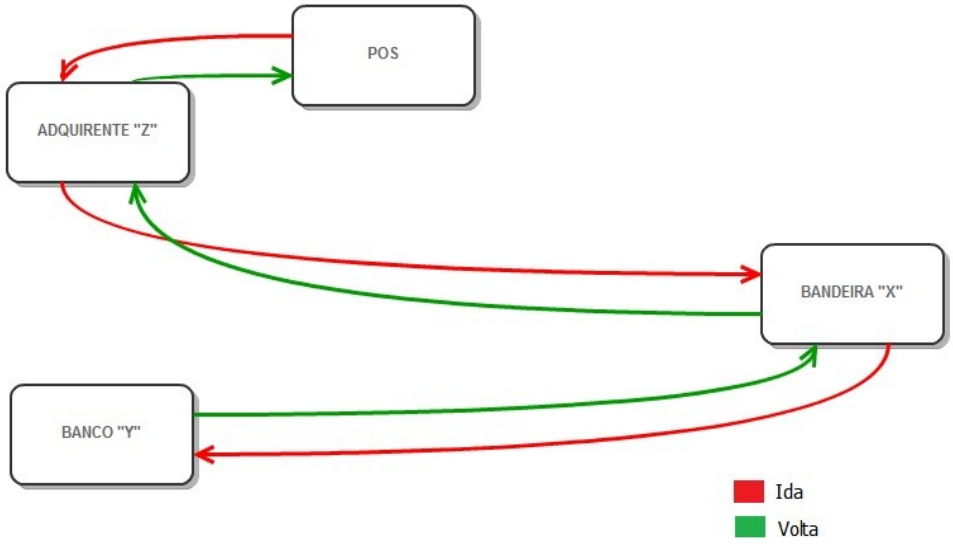

FIGURA 3.1 – FLUXO DE UMA TRANSAÇÃO / FONTE: O AUTOR.

A transação chega à bandeira “X” (normalmente os servidores das grandes bandeiras são localizados nos Estados Unidos) e esta identifica qual é o banco emissor daquele cartão através de um código identificador.

No exemplo acima, a bandeira “X” identificará e enviará a transação para os servidores do banco “Y”, onde o sistema autorizador fará validações relativas àquela compra (exemplo: limite disponível).

Se estiver tudo certo até esta etapa, o banco enviará para a bandeira uma autorização. A bandeira por sua vez devolverá aquela autorização para a adquirente e a mesma vai redirecioná-la ao terminal de onde a transação saiu. Neste momento aparece na tela do

terminal a mensagem "Aprovada". O fluxo é o mesmo para uma transação não aceita, a diferença estaria na mensagem final.

Se o cartão não possui *chip*, o início do fluxo não teria a primeira validação realizada no terminal POS (cartão válido e validação da senha). Essas validações seriam feitas somente pelo autorizador quando a transação chegasse aos servidores do banco.

Segundo ABECS (2010), cada bandeira e cada banco guardam um registro de todas as transações que passaram por ela e que foram autorizadas. Ao final de um prazo (normalmente de um ou dois dias), o banco recebe da bandeira um arquivo contendo um resumo dos registros. Então é feito uma comparação entre o que a bandeira enviou e o que o banco tem registrado. As compras só vão para a fatura do cliente se esse batimento acontecer, evitando possíveis transações que foram autorizadas pelo banco, mas se perderam no caminho de volta ao terminal POS.

3.2 - Impressão Digital

3.2.1 - Conceito

Quando se fala em impressão digital, refere-se a impressão resultante da reprodução da digital humana em uma superfície lisa, sendo esta corada por alguma substância, como a tinta (FERREIRA, 1999) ou capturada por leitores de impressão digital (ID) (DUARTE, 2004, p.3).

As digitais humanas são desenhos formados por dobras cutâneas das polpas dos dedos das mãos e dos pés. Localizam-se na camada da pele chamada derme e se reproduzem na epiderme, localizada logo acima, gerando infinitas configurações. É durante o período fetal, por volta do sexto mês de vida, que as impressões digitais são formadas.

As impressões mudam de tamanho, mas nunca de formato, permanecendo o mesmo pelo resto da vida, a não ser que sofram alterações causadas por fatores externos. Até mesmo gêmeos idênticos têm diferentes impressões digitais, e da mesma forma não podemos co-relacionar as impressões dos diferentes dedos de um indivíduo (PANKATI, 2000, p.3).

Como citado anteriormente, as IDs nunca mudam, exceto pela ação de fatores não naturais, como cortes profundos no dedo, amputações, lesões causadas pelo exercício de certas profissões ou queimaduras graves que resultem em desfiguração permanente da pele (FERREIRA, 1999).

A imagem do desenho digital também pode ser transferida para uma superfície através do toque do dedo quando as glândulas sudoríparas e sebáceas eliminam suor ou através de substâncias gordurosas que se encontram nas camadas subcutâneas e são eliminadas pelos poros que ficam na superfície das cristas papilares. Dessa forma, também podemos obter o registro perfeito dos desenhos digitais formados pelas papilas através da transferência dessas substâncias (TAVARES JUNIOR, 1991, p.30).

Segundo DUARTE (2004, p.3), o registro da impressão digital usando o método por leitores de IDs é o mais eficiente dentre os demais, pois utiliza sistema eletrônico de geração de dados, transformando aspectos físicos em um *template*, evitando assim a distorção dos dados por fatores como o excesso ou falta de tinta ou a intensidade com que se pressiona o dedo contra o papel.

3.2.2 - Datiloscopia

O nome datiloscopia é constituído de dois elementos gregos, *daktylos* = dedos e *Skopêlin* = examinar. Define-se datiloscopia como a ciência que estuda a identificação de pessoas pela comparação de impressões digitais, impressas em papel ou armazenadas em mídia magnética. Esse estudo envolve a verificação de pontos característicos da ID, conhecidos como minúcias (KEHDY, 1968, p.23). A datiloscopia apresenta três vertentes: civil, criminal e clínica.

A datiloscopia clínica estuda as perturbações nas IDs devido a ação de fatores como doenças ou exercício de profissões. A civil trata basicamente de autenticar a identidade de um indivíduo para acesso a recursos restritos, confecção de documentos, entre outros. A criminal, por sua vez, é utilizada para investigações e identificação de indivíduos suspeitos através da análise da cena do crime. (TAVARES JUNIOR, 1991, p.25).

Segundo KEHDY (1968, p.16) e TAVARES JUNIOR (1991, p.20), os estudos para aplicações de impressão digital como forma de identificação de indivíduos tiveram início por volta do ano 1664, com as pesquisas do anatomista italiano J. Marcello Malpighi. Essas pesquisas foram aperfeiçoadas por mais doze pesquisadores, destacando-se os seguintes:

1 William Hershel: em 1858 provou o postulado da imutabilidade;
2 Arthur Kollmam: em 1883 provou a formação dos desenhos digitais do sexto mês de vida fetal;
3 Francis Galton: em 1888 criou um sistema com trinta e oito tipos de IDs classificados em três grupos: arcos, presilhas e verticilios;
4 Henry de Varigny: em 1891 publicou um artigo sobre o sistema de Galton, sendo traduzido para várias línguas logo depois;
5 Juan Vucetich: leu o artigo de Varigny, criou uma extensão do sistema de Galton e implantou pela primeira vez a identificação de suspeitos de crime pela ID por volta de 1891.

Segundo KEHDY (1968, p.26), a eficiência dos sistemas baseados na análise do desenho digital e suas minúcias deve-se em parte aos quatro postulados da datiloscopia, definidos a seguir:

- Perenidade: indica que o desenho digital é capaz de durar muitos anos;
- Imutabilidade: dita a não mudança natural dos desenhos digitais desde o nascimento até a morte do indivíduo;
- Unicidade: afirma que nenhum dedo terá a digital igual a outro, variando de dedo para dedo e pessoa para pessoa. A unicidade é possível graças às minúcias;

- Classificabilidade: cita a possibilidade de classificar ou medir quantitativamente o desenho digital.

Nota-se, portanto, que o uso de IDs para identificação de pessoas vem sendo utilizado por dezenas de anos e sua veracidade tem sido bem estabelecida, aceita e comprovada.

3.2.3 - Minúcias das Impressões Digitais

As minúcias, ou detalhes de Galton, são acidentes que se encontram nas cristas papilares. É através destas minúcias que se estabelece a unicidade das IDs. As partes escuras, linhas pretas ou cinzas da ID são as cristas papilares e devem ser levadas em consideração em uma comparação. Por sua vez, os sulcos interpapilares são as linhas claras que separam as cristas papilares (COSTA, 2000, p.15).

Para KEHDY (1968, p.61) e TAVARES JUNIOR (1991, p.32), a confirmação da identidade de uma ID deve estar baseada na análise da impressão testemunha e impressão suspeita, a impressão encontrada no local de crime e a impressão tomada dos dedos do suspeito ou encontrada no arquivo datiloscópico, respectivamente. Na comparação entre essas duas digitais deverão ser coincididos no mínimo doze minúcias, as quais devem ser encontradas da mesma forma, localização e mesma quantidade. Não pode haver minúcias diferentes, que se encontram na impressão testemunha e não estão presentes na impressão suspeita. Isto também se aplica na identificação civil ou clínica.

As minúcias ou pontos característicos são resumidamente classificados dentro de duas categorias: os aspectos básicos e aspectos compostos. Os aspectos básicos são considerados as minúcias importantes, das quais se buscam informações para autenticar uma ID (COSTA, 2000, p.15).

Algum dos aspectos básicos mais comumente analisados são as cristas finais (CF), definidas como o ponto onde a crista termina e as cristas bifurcadas (CB) ou bifurcações,

definidas como um ponto onde a crista diverge dentro de cristas brancas ou onde a linha se divide em duas. Nas figuras 3.2 e 3.3 têm-se exemplos de CF e CB, respectivamente.

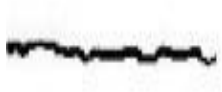

FIGURA 3.2 - CRISTA FINAL / FONTE: DUARTE (2004, P.5).

FIGURA 3.3 - CRISTA BIFURCADA / FONTE: DUARTE (2004, P.5).

Da mesma forma, algum dos aspectos compostos mais comumente analisados são as ilhas, cruzamentos, esporas e cristas curtas, como representado na figura 3.4. As ilhas ou lagos são formados por duas bifurcações conectadas, que se contornam e retornam ao rumo de origem. Cristas curtas são definidas como CFs muito pequenas. Esporas são formadas pela combinação de CBs e CFs. Cruzamentos ou pontes são definidos como duas ou mais bifurcações com um caminho conectando-as.

FIGURA 3.4 – EXEMPLO DE ASPECTOS COMPOSTOS DE MINÚCIAS / FONTE: KEHDY, 1968, P.150

Existem ainda mais dois tipos de minúcias usadas para classificar uma ID: o núcleo, que é a volta mais interna do conjunto das cristas; e o delta, que corresponde ao maior ângulo ou a um triângulo isósceles entre as cristas. Essas minúcias podem ser encontrados em uma ID em diferentes posições, tamanhos e quantidades, podem possuir um núcleo e um delta, ou então um núcleo e dois deltas, ou ainda por cima, podem não existir. Na figura 3.5 tem-se um exemplo de um ID com um núcleo e dois deltas.

FIGURA 3.5 – EXEMPLO DE NÚCLEOS E DELTAS / FONTE: MAZI, DAL PINO JÚNIOR, 2009, P. 2

3.2.4 - Sistema de Vucetich – Tipos de Impressões Digitais

Os deltas, mostrados na seção anterior, são responsáveis pelos sistemas de classificação de ID existentes. O delta determina a classe da ID dependendo de sua posição na impressão, sendo também responsável pelos contornos que formam o núcleo da ID. Na ausência do delta não se encontra um núcleo definido, tendo somente a região marginal (ponta do dedo) e a região basilar (início do desenho digital). Quando da existência de um ou dois deltas tem-se as duas regiões, marginal e basilar, incluindo um núcleo (define o centro da impressão). Estas três nomenclaturas de divisões de partes da impressão digital são denominadas de sistema de linhas como mostrado na figura 3.6 (KEHDY, 1968, p.30).

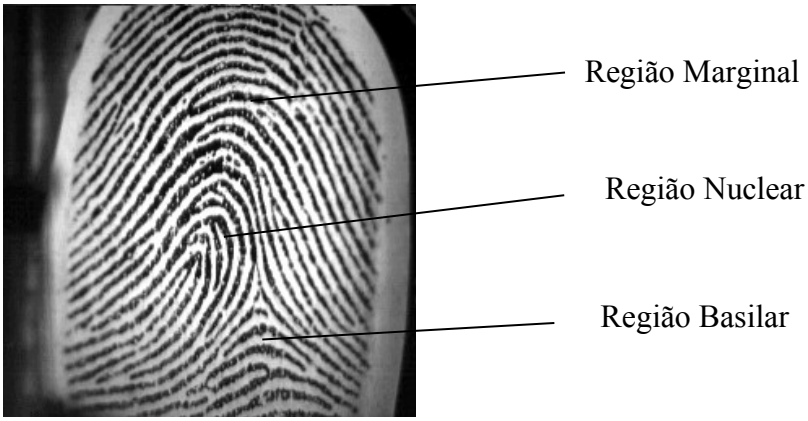

FIGURA 3.6 – SISTEMA DE LINHAS / FONTE: O AUTOR

KEHDY (1968, p.32) e TAVARES JUNIOR (1991, p.37) afirmam que no Brasil, o sistema datiloscópico utilizado é o idealizado pelo pesquisador Juan Vucetich. Nesse sistema de classificação, existem quatro tipos de ID, cuja denominação varia de acordo com a posição e quantidade de deltas:

- Arco (possivelmente sem delta; atravessam de um lado ao outro assumindo uma forma convexa);
- Presilha interna (delta a direita do observador; as linhas nucleares dirigem-se à esquerda);
- Presilha externa (delta a esquerda do observador; as linhas nucleares dirigem-se à direita);
- Verticilo (dois deltas, um a direita e outro a esquerda; as linhas nucleares assumem configurações variadas).

Encontra-se no sistema de classificação de Vucetich mais quatro subtipos que são agregados as quatro classes principais, sendo esta subclassificação baseada na configuração do núcleo. São eles:

- Arco: plano, angular, bifurcado à direita e bifurcado à esquerda;
- Presilha interna: normal, invadida, dupla e ganchosa;

- Presilha externa: normal, invadida, dupla e ganchosa; e
- Verticilo: espiral, ovoidal, sinuoso e ganchoso.

Na figura 3.7 são exemplificados os tipos mais comuns de IDs encontrados:

FIGURA 3.7 – TIPOS FUNDAMENTAIS DE IMPRESSÕES DIGITAIS / FONTE: KEHDY, 1.968, P.32.

De acordo com KEHDY (1968, p.35) os quatro tipos fundamentais não aparecem na mesma proporção. Estudos estabeleceram que os tipos de ID são distribuídos desigualmente, aproximadamente da seguinte forma:

- 60% do tipo presilha;
- 35 % do tipo verticilos;
- 5% do tipo arco.

Ainda segundo KEHDY (1968, p.36), existem IDs irregulares ou anormais, conhecidas como anômalas, que apresentam desenhos considerados inclassificáveis. Essas falhas podem ser adquiridas ou congênitas.

Algumas das anomalias congênitas mais comuns e que dificultam o enquadramento da ID em um determinado tipo são (TAVARES JUNIOR, 1991, p.43):

- Sindactilia: dedos unidos pela pele;

- Polidactilia: mais de cinco dedos nas mãos ou pés;
- Ectrodactilia: menos de cinco dedos;
- Macrodactilia: dedos maiores que o normal;
- Microdactilia: dedos menores que o normal.

Existem ainda IDs que apresentam seu desenho encaixando-se em mais de um tipo, chamados de tipos limites, eles podem ser (KEHDY, 1968, p.36):

- Arco e presilha interna;
- Arco e presilha externa;
- Arco e verticilo;
- Presilha interna e verticilo e;
- Presilha externa e verticilo.

3.3 - Biometria

3.3.1 - Definição

Segundo PINHEIRO (2008), podemos definir a biometria como a ciência da aplicação de métodos de estatística quantitativa a fatos biológicos, ou seja, é o ramo da ciência que se ocupa da medida dos seres vivos (do grego *bio* = vida e *métron* = medida). A biometria reconhece um indivíduo pelas suas características humanas mensuráveis e assim autentica a sua identidade.

Por outro lado, quando nos referimos à biometria, estamos tratando das tecnologias que analisam essas características para fins de segurança. Por definição “a biometria é uma característica única mensurável ou um traço do ser humano que automaticamente reconhece ou verifica sua identidade” (CBA, 2010).

As tecnologias biométricas, dessa forma, fazem referencia às partes físicas e as características pessoais e únicas dos seres humanos, devendo reconhecer de forma rápida e automática, de preferência em tempo real tais características (CBA, 2010).

HONG (1998, p.4) afirma que para o desenvolvimento de sistemas de identificação biométricos é necessário analisar características físicas e comportamentais que são ligadas a alguns conceitos muito importantes devendo:

- Ser tão únicas quanto possível, ou seja, um traço idêntico jamais aparecerá em duas pessoas: singularidade;
- Existir em tantas pessoas quanto possível: universalidade*;*
- Ser medida com instrumentos técnicos simples: mensurabilidade*;*
- Ter suas características imutáveis: permanência;
- Ser fáceis e confortáveis de serem medidas: uso amigável.

Muito embora na prática nem todos os sistemas biométricos possam atender a todos estes critérios, existem outros aspectos como aceitabilidade, velocidade, performance e impostura que determinam o desempenho de um sistema biométrico.

"Um sistema de identificação é essencialmente um sistema de recuperação de dados onde a exatidão e recuperação dos dados também são medidas de precisão" (PANKATI, 2000, p.5).

3.3.2 - Histórico da Biometria

De uma maneira diferente e sem a sofisticação que encontramos hoje, a biometria já existe há séculos. A análise das características humanas vem sendo usada durante toda a História como um modo de identificação. Desde crianças aprendemos a lembrar de alguém ou associar certas características como o rosto ou o som da voz a uma determinada pessoa. Da mesma forma, a assinatura é usada há séculos como método de autenticação e reconhecimento (CBA, 2010).

Ainda de acordo com CBA (2010), o estudo do que hoje chamamos de biometria teve início com um cientista chamado Francis Galton. Sua pesquisa foi pioneira na aplicação de métodos estatísticos para fenômenos biológicos.

Em 1884, Galton abriu o Laboratório de Antropométrica na Exibição Internacional de Saúde, onde coletou estatísticas de milhares de pessoas. Logo depois, em 1892, inventou o primeiro sistema moderno de impressão digital. Adotado pelos departamentos de polícia em todo o mundo, a impressão digital era a forma mais confiável de identificação, até o advento da tecnologia do DNA no século XX (CBA, 2010).

Na década de 1970, com o advento do sistema Identimat, que foi instalado em diversos locais secretos para controle de acesso, a área da biometria começou a ter avanços comerciais. Esse sistema mensurava a forma da mão e analisava principalmente o tamanho dos dedos. A produção do Identimat acabou na década de oitenta e seu uso é considerado pioneiro na aplicação da geometria da mão e para a tecnologia biométrica como um todo (CBA, 2010).

CBA (2010) afirma que na mesma época, o estudo da biometria digital teve início dada a necessidade de automatizar a identificação das imagens digitais, auxiliando assim as forças policiais, uma vez que a comparação manual das imagens digitais com os registros criminais era longa e demandava muito trabalho.

Já no final da década de 1960, o FBI começou a checar as imagens digitais automaticamente e na metade da década de setenta já havia instalado uma boa quantidade de sistemas de *scanners* digitais automáticos. A partir daí, o papel da biometria nas forças policiais em todo o mundo tem crescido rapidamente e os *Automated Fingerprint Identification Systems* (AFIS) são cada vez mais utilizados.

As pesquisas em universidades e o investimento por parte da indústria fomentam a contínua comercialização desse tipo de tecnologia, criando novas técnicas que, depois de levadas ao mercado e de provadas seu desempenho operacional, são submetidos às mais variadas aplicações militares e civis (academias, universidades, empresas ou prédios comerciais) (REVISTA DIGITAL, 2009, p. 7).

3.3.3 - Identificação, Reconhecimento e Verificação

Identificação, reconhecimento e verificação são termos comuns em sistemas biométricos e precisam ser diferenciados e bem definidos dados a sua importância para o entendimento do funcionamento de tais sistemas.

De acordo com CBA (2010) e ASHBOURN (2000), identificação e reconhecimento podem ser agrupados. Aqui, um determinado exemplo é apresentado ao sistema biométrico durante um cadastramento. O sistema então tenta encontrar a quem esse exemplo pertence, através da comparação do exemplo com um banco de dados objetivando encontrar um par.

Por outro lado, a verificação é um processo um-para-um (1:1), onde o sistema biométrico tenta verificar a identidade do indivíduo. Aqui um único exemplo biométrico é comparado com outro exemplo. Uma característica biométrica é recolhida. Futuramente um novo exemplar desta característica será recolhido e comparado com o exemplar previamente cadastrado. Se os dois combinarem, o sistema confirmará que a pessoa é quem diz ser (CBA, 2010).

Resumindo, a diferença fundamental é que a identificação e o reconhecimento trabalham com um exemplo contra um banco de dados (1:N), enquanto a verificação trabalha a comparação de um exemplo com outro (1:1). A identificação pergunta “Quem é esse?”. A verificação, por sua vez, pergunta “É esta pessoa quem ela diz ser?” (CBA, 2010).

3.3.4 - Como Funciona a Biometria

3.3.4.1 - O Procedimento

Segundo CBA (2010) e ASHBOURN (2000), todos os métodos de autenticação biométrica funcionam de maneira parecida. Primeiramente, durante um processo de cadastramento, o sistema precisa capturar um exemplo da característica a ser analisada.

Atributos únicos são então extraídos e convertidos pelo sistema em um código matemático. Esse exemplo é então armazenado como o *template* biométrico daquela pessoa. O *template* pode ser armazenado em um sistema biométrico, um banco de dados, um cartão inteligente, um código de barras ou em qualquer outra forma de memória de armazenamento.

Outras técnicas podem ser aplicadas conjuntamente, por exemplo, pode haver um gatilho, ou alguma maneira de ligar o *template* armazenado àquela pessoa. Pode ser também que um cartão com o *template* armazenado seja colocado num leitor de cartões. Independente da técnica utilizada, o usuário final necessita interagir com o sistema biométrico por uma segunda vez para ter sua identidade checada. Um novo exemplo biométrico é então tirado e comparado com o *template*. Se o *template* e o novo exemplo combinarem, o usuário ganha o acesso. Este é um conceito fundamental de um sistema biométrico: uma pessoa precisa ter um exemplo do dado biométrico capturado e comparado pelo sistema para que ele decida se ele combina com o exemplo previamente armazenado (CBA, 2010).

CBA (2010) e HONG (1998, p.7) ressaltam que apesar de muito confiáveis, devemos notar que nenhum sistema biométrico garante 100% de precisão. Os seres humanos são inconsistentes e tanto as características físicas quanto as comportamentais podem mudar com o tempo. Além disso, o modo do ser humano interagir com uma máquina pode não ser constante. Fatores como *stress*, saúde geral, trabalho, condições ambientais e pressões do tempo podem ocasionalmente fazer com que os seres humanos tornem-se instáveis.

Os sistemas biométricos devem permitir tais mudanças. Em outras palavras, se o exemplo biométrico é suficientemente similar ao *template* previamente armazenado, o sistema determinará que as duas de fato combinam. Se não, o sistema não encontrará um par e não identificará o usuário. Este conceito dá à tecnologia biométrica uma vantagem significante sobre as senhas, PINs e cartões de identificação, pois estas não apresentam grande grau de flexibilidade (CBA, 2010).

3.3.4.2 - Procedimentos Operacionais

CBA (2010) afirma que todos os sistemas biométricos utilizam os quatro passos de procedimento: captura, extração, comparação e combinação. No centro do sistema biométrico reside um elemento proprietário - a máquina - a qual extrai e processa o dado biométrico. Para isso pode-se utilizar um algoritmo ou uma rede neural artificial. Extrai-se o dado, cria-se um *template* e computam-se os dados do *template* e do novo exemplo. A figura 3.8 mostra como se seguem os passos do procedimento.

- Captura - Um exemplo físico ou comportamental é capturado pelo sistema durante a fase de cadastramento;
- Extração - Um dado único é extraído do exemplo e um *template* é criado;
- Comparação - O *template* é comparado com um novo exemplo;
- Combinação/Não-Combinação - O sistema decide se o atributo extraído do novo exemplo constitui um par ou não.

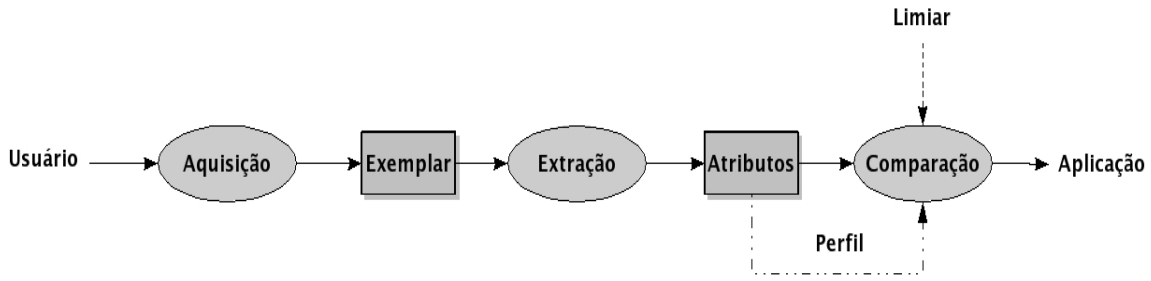

FIGURA 3.8 - PASSOS DO PROCEDIMENTO BIOMÉTRICO / FONTE: OBELHEIRO, COSTA, FRAGA, 2010

3.3.5 - Falsa Rejeição e Falsa Aceitação

Analisando a capacidade do sistema de permitir o acesso a usuários autorizados e negar o acesso a usuários não autorizados, há alguns anos a indústria biométrica tem utilizado duas medidas de desempenho para graduar o nível de precisão dessa comparação. São

conhecidas como a Taxa de Falsa Rejeição (FRR) ou taxa de erro do Tipo I e Taxa de Falsa Aceitação (FAR) ou taxa de erro do Tipo II (WAYMAN,1999).

A taxa de falsa rejeição refere-se à quantidade de vezes que um indivíduo autorizado é falsamente rejeitado pelo sistema. A taxa de falsa aceitação, pelo contrário, refere-se à quantidade de vezes que um indivíduo não-autorizado é falsamente aceito pelo sistema (CBA, 2010).

Ambas as taxa são expressas na forma de porcentagem, utilizando-se os cálculos que seguem abaixo:

$$FRR = (\frac{NFR}{NTV})100 \tag{3.1}$$

Onde, NFR = Número de falsas rejeições e NTV = Número de reconhecimentos autorizados ou tentativas de verificação.

$$FAR = (\frac{NFA}{NTV})100 \tag{3.2}$$

Onde, NFA = número de falsas aceitações e NTV = número de reconhecimentos autorizados ou tentativas de verificação.

Menos usada como referencial de análise, a taxa de erros (EER) é obtida exatamente no ponto onde a FRR e a FAR se encontram quando colocadas em um gráfico, como representado na figura 3.9. Por exemplo, um sistema com uma FRR e uma FAR de 1% também terá um EER de 1%. Através desta comparação se permite o balanceamento de conveniência x segurança que se pretende dar ao sistema (OBELHEIRO, COSTA, FRAGA, 2010).

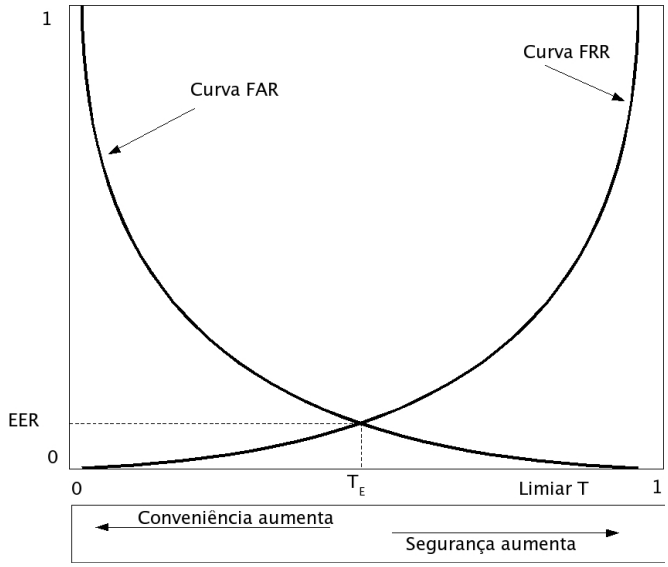

FIGURA 3.9 - GRÁFICO FAR X FRR / FONTE: OBELHEIRO, COSTA, FRAGA, 2010

Para CBA (2010), o uso de taxas de falsa rejeição e falsa aceitação tem causado grandes controvérsias dentro da indústria biométrica. Essas discussões referem-se principalmente à significância estatística de cálculos tão simplificados. As fórmulas acima são surpreendentemente simples, dado o grande número de variáveis que pode ser apresentado ao sistema biométrico. O desempenho de um sistema biométrico, a FRR e a FAR podem ser afetadas por:

- Condições ambientais: por exemplo, temperaturas extremas e umidade;

- A idade, o sexo, a etnia e a ocupação do usuário: por exemplo, mão sujas de trabalhos manuais;
- As crenças, desejos e intenções do usuário: se um usuário não quiser interagir com o sistema, o desempenho será afetado;
- A aparência física do usuário: um usuário sem membros não poderá usar biometrias baseadas nos dedos ou na mão.

Verificando a taxa de erros e considerando a FRR e a FAR em concordância é fácil determinar o desempenho básico do sistema, sempre lembrando que as circunstâncias de uma aplicação não devem ser esquecidas, especialmente se um sistema biométrico pretende operar dentro dos limites de uma certa aplicação (CBA, 2010).

Para uma análise mais delicada de comparadores, existem outros conceitos úteis como a separação das densidades de probabilidade e o conceito de Erro Total Esperado, com seu refinamento associado a Funções de Custo para cada tipo de erro. Estas Funções de Custo levam em consideração a vocação do sistema e a criticidade dos dados e ativos manipulados pelo sistema (OBELHEIRO, COSTA, FRAGA, 2010).

3.3.6 - Tipos de Autenticação Biométrica

Para PINHEIRO (2008), existem várias características biológicas que podem ser usadas em um processo de identificação. Vejamos as principais:

- Íris: baseado na leitura da íris, anel colorido que circunda a pupila do olho. Cada íris possui uma estrutura única, caracterizando um padrão complexo. Pode ser uma combinação de características específicas como coroa, glândula, filamentos, sardas, sulcos radiais e estriamentos (PINHEIRO, 2008);

- Retina: baseado na leitura da retina, camada de veias sangüíneas situada na parte de trás do olho. Assim como na íris, a retina forma um padrão único e começa a se desintegrar logo após a morte (PINHEIRO, 2008);

- Face: baseado na análise do indivíduo através da leitura da face, um processo complexo que normalmente requer artifícios inteligentes sofisticados e técnicas de aprendizagem computacional (*machine learning techniques*). A inteligência artificial é necessária para simular a interpretação humana das faces, se adaptando às diversas mudanças que ocorrem com estas e para comparar precisamente os novos exemplos com os *templates* previamente armazenados (PINHEIRO, 2008);

- Geometria da Mão: a biometria de geometria da mão tira uma imagem tridimensional da mão e mede o seu tamanho, o comprimento dos dedos e das articulações (PINHEIRO, 2008);

- Geometria do Dedo**:** baseado na análise das medidas de atributos únicos do dedo, como a largura, comprimento, espessura e o tamanho das articulações. Usa princípios similares aos da geometria da mão (PINHEIRO, 2008);

- Palma: a biometria da palma pode ser estreitamente associada com a impressão digital e particularmente com a tecnologia AFIS. Os dados das linhas papilares, vales e minúcias são encontrados na palma, assim como nas imagens digitais. Normalmente eles são analisados usando técnicas de captura óptica (PINHEIRO, 2008);

- Assinatura: a biometria de assinatura geralmente é denominada como uma Verificação Dinâmica de Assinatura (DSV) e observa a forma como assinamos nossos nomes. É a forma de assinar, mais do que a assinatura acabada, que realmente importa. Por exemplo, o ângulo no qual a caneta é segurada, o tempo que se leva para assinar, a velocidade e a aceleração da assinatura, a pressão exercida quando segura-se a caneta e o número de vezes que a caneta é levantada do papel. Tudo isso pode ser extraído como características comportamentais únicas (PINHEIRO, 2008);

- Voz: essas biometrias estão focalizadas no som (timbre) da voz. A técnica de medição da voz pode usar tanto métodos dependentes de texto ou não. Ou seja, a voz pode ser capturada com um usuário falando uma senha específica de frases combinadas, palavras ou números (dependente), ou qualquer forma de frase, palavras ou números (independente). Particularmente útil para aplicações baseadas na telefonia (PINHEIRO, 2008);

- Digital: A mais estudada característica biométrica. Por serem únicas para cada indivíduo, as impressões digitais são consideradas o tipo biométrico mais seguro para determinar a identidade, depois do teste do DNA. Baseado na identificação através das irregularidades das impressões digitais, retiradas de um ou mais dedos, analisando as

minúcias conforme descrito anteriormente. É necessário para este tipo de reconhecimento o uso de um dispositivo capaz de capturar as características da digital humana, além de um *software* que trate a imagem e faça um posterior reconhecimento e comparação da digital. É o tipo biométrico mais utilizado em todo mundo, pouco invasivo ao usuário e de baixo custo (PINHEIRO, 2008).

3.3.7 – Comparativo

Cada um dos fatores a seguir tem sido separado pela simplicidade e estão graduados como baixo, médio, alto ou muito alto. São usados como parâmetros para decisão de qual característica biométrica se irá usar em um determinado projeto ou sistema (CBA, 2010):

- Nível de Precisão: Qual é a precisão geral da biometria?

- Facilidade de Uso: Instruções especiais são necessárias? Quão fácil é para se usar efetivamente a biometria?

- Barreiras ao Ataque: Quão boa é a biometria na prevenção de acessos fraudulentos?

- Aceitabilidade do Público: Quão confortável o público se sente com a biometria?

- Estabilidade a Longo Prazo: As características físicas ou comportamentais mudam com o tempo?

E também:

- Padrões: Existem padrões? Os padrões estão sendo desenvolvidos?

- Interferência: Quais fatores podem interferir potencialmente no processo de captura?

Pode-se observar nos quadros 3.1 e 3.2 um comparativo simplificado das tecnologias de autenticação biométrica, de maneira que se possa determinar a melhor opção.

QUADRO 3.1 - COMPARATIVO ENTRE BIOMETRIAS

	Nível de Precisão	Facilidade de Uso	Barreiras ao Ataque	Aceitabilidade do Público	Estabilidade à Longo Prazo
Iris	Muito Alta	Média	Muito Alta	Média	Alta
Retina	Muito Alta	Baixa	Muito Alta	Média	Alta
Face	Alta	Média	Média	Alta	Média
Impressão Digital	Alta	Alta	Alta	Média	Alta
Geometria da Mão	Alta	Alta	Alta	Alta	Média
Geometria do Dedo	Alta	Alta	Alta	Média	Média
Palma	Alta	Alta	Alta	Média	Alta
Assinatura	Alta	Alta	Média	Muito Alta	Média
Voz	Alta	Alta	Média	Alta	Média

FONTE: ADAPTADO DE CBA(2010)

QUADRO 3.2 - COMPARATIVO ENTRE PADRÕES DE BIOMETRIAS

	Padrões	Interferência
Iris	-	Óculos de grau ou escuro
Retina	-	-
Face	-	Iluminação fraca; envelhecimento da face; óculos; pêlos faciais
Impressão Digital	ANSI/NIST Data Interchange & FBI Image Compression Standards	Umidade, sujeira ou imagens digitais danificadas; idade; sexo e raça do usuário final
Geometria da Mão	-	Doenças como artrite e reumatismo nos usuários finais
Geometria do Dedo	-	Doenças como artrite e reumatismo nos usuários finais
Palma	ANSI/NIST Data Interchange & FBI Image Compression Standards	Umidade, sujeira ou imagens digitais danificadas; idade; sexo e raça do usuário final
Assinatura	-	Falta de instrução; assinaturas que mudam constantemente ou facilmente copiáveis
Voz	Speaker Verification API (SVAPI)	Barulhos no ambiente; resfriados e outros fatores que modificam a voz

FONTE: ADAPTADO DE CBA(2010)

As vantagens e a grande aplicabilidade das biometrias baseada na impressão digital são evidentes. Além disso, o custo-benefício em relação aos fatores acima descritos justificam a escolha dessa biometria para o desenvolvimento do projeto. Trata-se de uma tecnologia em constante expansão, sendo usada nos principais sistemas biométricos do mercado. De acordo com a pesquisa do Grupo Internacional de Biometria, os sistemas biométricos baseados em digitais compreendem 66,7% do total.

3.4 - *Smart Cards*

Os cartões *smart card* vem sendo amplamente utilizado nos dias de hoje, principalmente por instituições financeiras e empresas de telefonia.

Segundo ABNT (2011), esses cartões oferecem mais segurança e confidencialidade do que qualquer outro veículo de informação financeira ou armazenamento de transações. As informações armazenadas em um *smart card* são guardadas em um micro *chip*, que pode ser seguro de várias maneiras. Primeiro por ter a capacidade de ter suas próprias chaves de segurança, tais como códigos PIN. Segundo, uma vez que o *chip* está envolto pelo cartão, é praticamente impossível violar suas informações sem causar danos permanentes ao plástico ou ao próprio *chip*. Por último, o cartão com *chip* elimina em muitas aplicações a necessidade de digitação de informações que podem ser vistas por terceiros.

Para SOUZA (2010), outro benefício destes cartões é a proteção aos dados inseridos em suas memórias através de algoritmos criptográficos que blindam as informações, dificultando a interceptação ou extração. Algoritmos de criptografia simétrica como DES 34 e DES Triplo e de chave assimétrica como o RSA, são exemplos de funcionalidades criptográficas presentes em muitos *smart cards*, tendo em vista que cada fabricante define qual algoritmo será implantado em seu cartão, bem como a chave criptográfica.

Além disso, a clonagem de cartões, prática muito comum no ambiente financeiro, torna-se muito mais trabalhosa e onerosa, chegando ao ponto de inibir a ação de bandidos devido aos gastos e empenhos dedicados para clonar ou capturar a informação armazenada em um *chip* de *smart card* (SOUZA, 2010).

3.4.1 - Evolução dos *smart cards*

As primeiras experiências com inserção de circuitos integrados em cartões datam do final da década de 70. Na Alemanha e no Japão constam registros de invenções relacionadas a esta tecnologia. Porém, foram com as patentes registradas pelo inventor Roland Moreno entre os anos de 1974 e 1979, que a fabricação em larga escala dos cartões com *chip* se deu início.

Em 1984, na França, com a ideia de substituir as fichas pelos cartões nos telefones públicos, teve início a primeira aplicação com *smart cards*. Logo depois, na Alemanha, a mesma aplicação foi implementada.

Depois disso, com a expansão de suas aplicações, surgiu a necessidade de reutilização do cartão, bem como de se acrescentar alguma capacidade de processamento a eles (ABNT, 2011).

3.4.2 - Classificação

A classificação de um cartão de circuito integrado se dá através da análise de diversos fatores. Segundo ABNT (2011), os fatores de classificação são os seguintes:

Com relação ao contato:

- Cartões com contato: o acesso aos dados armazenados se dá através do contato físico entre o *chip* do cartão e um dispositivo de leitura/gravação. O cartão e o dispositivo se comunicam através de sinais elétricos. Nesse tipo de cartão o microprocessador fica aparente possibilitando o seu contato e leitura;

- Cartões sem contato: nesse tipo de cartão não há contato físico entre o microprocessador e o dispositivo de leitura/gravação. A leitura pode ocorrer de diversas formas e a distância entre o cartão e o dispositivo de leitura/gravação pode variar de milímetros até metros de distância, inclusive sem a necessidade de se permanecer parado em relação ao dispositivo. A comunicação pode ocorrer através de infra-vermelho, indutância magnética, sinais de rádio-frequência, entre outros.

De maneira geral, as transações financeiras exigem a utilização de cartões de contato devido a sua segurança e confiabilidade.

Com relação as funções do *hip*:

- Cartão de memória: nenhum tipo de processamento ocorre nesses cartões. Toda a lógica de processamento está armazenada na leitora. Servem apenas para armazenamento de informações. Podem ser reutilizáveis ou não;

- Cartão microprocessado: são cartões capazes de realizar processamentos e executar comandos, com área de CPU e de memória. São os verdadeiros "cartões inteligentes".

Os cartões microprocessados se dividem em:

- Cartões Mono Aplicação: são concebidos para executar apenas um único serviço. Podem através do seu sistema operacional executar várias transações;

- Cartões Multi Aplicação: possuem sistema operacional e uma estrutura de diretórios, gerenciando assim suas áreas de memória. As aplicações financeiras requerem esse tipo de cartão.

Existem ainda os cartões híbridos que possuem além do *chip*, tarja magnética ou banda óptica.

3.4.3 - Produção de um *smart card*

Antes de serem inseridos no plástico do cartão, os *chips* são montados sobre áreas de contato e envolvidos por uma superfície protetora. Ainda durante esse processo pode ocorrer o processo de inicialização do *chip*, que nada mais é do que formatá-lo de acordo com a configuração necessária para aplicação fim (ABNT, 2011).

Logo depois, a máscara ou conjunto de estruturas e arquivos é feito de acordo com a aplicação necessária, em função do negócio a que se destina.

Uma vez concluídos esses processos, o *chip* é incorporado ao plástico. Na fase final o cartão pode ser personalizado, recebendo a aplicação, os dados do emissor e posteriormente os dados do cliente.

3.4.4 - Leitoras/Gravadoras de *Smart Card* ou Dispositivos de Aceitação

São dispositivos capazes de se comunicar com o cartão *smart card.* Segundo ABNT (2011), podem ser classificadas de acordo os seguintes fatores:

Quanto a capacidade de processamento:

- Leitoras/Gravadoras Escravas: podem apenas se comunicar com o cartão e energizá-lo, sem a capacidade de processar a aplicação. Precisam ser conectadas a um micro ou POS;

- Leitoras/Gravadoras *Stand-Alone*: possuem capacidade de processamento, suportando algumas aplicações. Podem trabalhar *off-line* ou em rede. Para aumentar sua segurança, a arquitetura é baseada em um processador de segurança como o SAM (*Secure Application Module*).

Quanto aos tipos de cartão que lê e grava:

- Leitoras/Gravadoras Dedicadas: são capazes de ler apenas cartões *smart card* (com ou sem contato);

- Leitoras/Gravadoras Híbridas: lêem cartões com tarja magnética e com *chip.*

Existem ainda as leitoras com motor, capazes de tracionar o cartão durante a inserção e liberação, facilitando assim o processo de leitura, bem como as leitoras projetadas para ATMs e PDVs.

3.4.5 - Componentes de um *smart card*

Segundo ABNT (2011), todos os padrões referentes aos *smart cards* de contato são estabelecidos através do ISO 7816. Esse ISO se divide em oito subitens, conforme a seguir:

- ISO 7816-1: inclui itens como força mecânica e resistência elétrica dos contatos, perfil da superfície dos contatos e interferência eletromagnética, entre outros;

- ISO 7816-2: determina o padrão e localização dos contatos;

- ISO 7816-3: determina os sinais elétricos e os protocolos de comunicação;

- ISO 7816-4: determina as estruturas de diretórios e a padronização dos comandos de acesso;

- ISO 7816-5: determina o sistema numérico e os procedimentos de registro para identificação da aplicação;

- ISO 7816-6: é um esboço do padrão internacional a ser implementado, determinando os elementos de dados industriais;

- ISO 7816-7 e 7816-8: melhorias em comandos e funções de segurança. Em estágio inicial.

Segundo CHEN (2000), o *chip* do cartão *smart card* possui oito pontos de contato, sendo definidos da seguinte forma:

- VCC: Responsável pelo fornecimento de energia ao *chip*. Pode variar de 3 a 5 Volts;

- RST: Utilizado para reinicialização do microprocessador através do envio de um sinal;

- CLK: Fornecimento do sinal de *clock* externo. O *clock* interno é estabelecido a partir do valor de referência do *clock* externo;

- GND: Utilizado para estabelecer a tensão de referência. Possui valor de 0 volts como padrão;

- VPP: Utilizado para alteração da voltagem durante a programação. Apenas cartões antigos possuem este ponto;

- I/O: Responsável pela transmissão de dados e comandos entre o cartão e o meio externo. Pode receber e enviar dados, mas não de forma simultânea;

- RFU: Pontos criados para possíveis aplicações futuras.

A figura 3.10 ilustra a disposição dos pontos de contato dentro do *chip.*

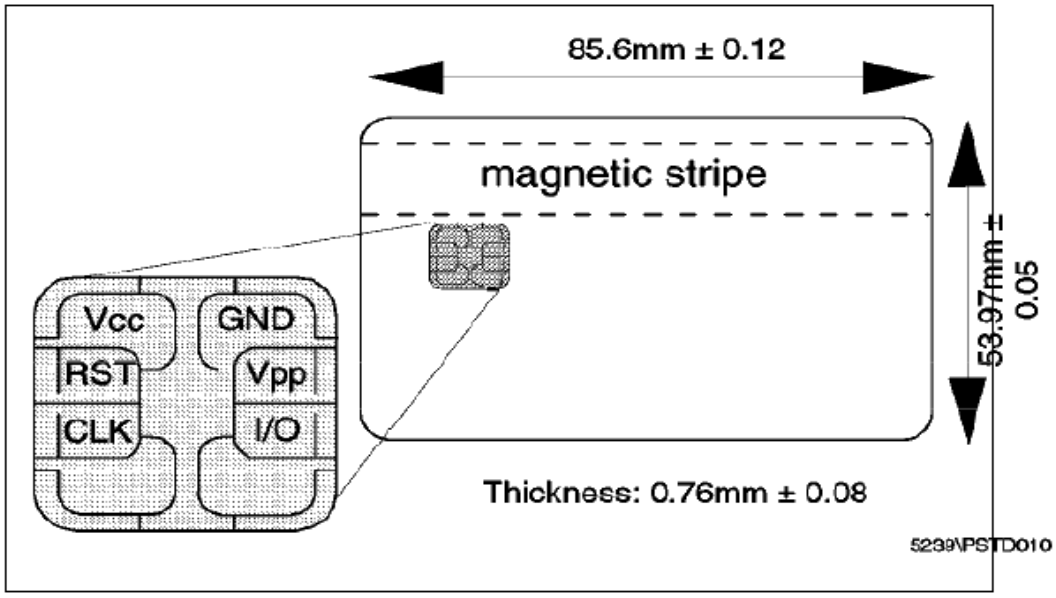

FIGURA 3.10 – APARÊNCIA FÍSICA DE UM SMART CARD / FONTE: CHEN, 2000

A UCP (Unidade Central de Processamento) de um *smart card* normalmente contempla um microcontrolador de 8 *bits*, podendo chegar a 16 ou 32 *bits* nos cartões mais modernos. A velocidade de *clock* geralmente é de 5MHz, podendo chegar até 40MHz nos cartões atuais (CHEN, 2000).

Com relação a memória dos cartões *smart card*, para CHEN (2000) podemos classificá-las em três tipos:

- ROM: Por não ser volátil e não necessitar de energia para acesso a seus dados, é usada para armazenar os programas fixos do cartão. O seu conteúdo não pode ser alterado;

- EEPROM: Semelhante a memória ROM, porém com a capacidade de ter seus dados alterados durante a utilização do cartão. É nesse tipo de memória que os dados do usuário são gravados;

- RAM: Usado para modificação e alocação de dados, é uma memória volátil e seus dados permanecem apenas enquanto existe fornecimento de energia.

As novas tecnologias envolvendo memórias *flash* deverão ser incorporadas aos *smart cards* em um futuro próximo.

3.4.6 - Protocolos de Transmissão e Comunicação

A comunicação dos cartões s*mart card* com o meio externo é baseada em dois conceitos fundamentais. O primeiro é que a comunicação é do tipo *half-duplex*, ou seja, o cartão tem a capacidade de enviar e receber mensagens, porém não ao mesmo tempo. O segundo conceito refere-se ao seu relacionamento com os dispositivos externos que segue o modelo mestre-escravo, no qual o cartão é o escravo, aguardando comandos provenientes de tais dispositivos (CHEN, 2000).

Os comandos recebidos pelo cartão obedecem a um protocolo da camada de aplicação chamado APDU. Esse protocolo basicamente é responsável por prover os serviços entre o cartão e o computador.

As mensagens APDU podem ter duas estruturas diferentes, uma para transmissão de dados entre o cartão e o computador e outra para transmissão entre o computador e o cartão (CHEN, 2000).

Cada campo de uma mensagem APDU é definido da seguinte maneira:

- CLA: Classe de instrução. Identifica a categoria de comando e resposta de um APDU;
- INS: Código de instrução. Especifica a instrução do comando;
- P1 e P2: Parâmetros utilizados para prover qualificações para a instrução;
- LC: Especifica o tamanho do campo de dados. É opcional;
- DATA FIELD: Contém os dados que serão enviados ao cartão para serem executados conforme as instruções dadas;
- LE: Número de *bytes* da resposta enviada pelo cartão ao computador.

Ainda segundo CHEN (2000), nas mensagens de resposta temos:

- DATA FIELD: Contém os dados da resposta. Seu tamanho deve estar de acordo com o especificado no campo LE da mensagem de comando;
- SW1 e SW2: Formam juntos a palavra de status. O status informa se o comando foi executado corretamente ou se ocorreu algum erro.

A figura 3.11 ilustra os campos de uma mensagem APDU.

COMANDO APDU						
CABEÇALHO (OBRIGATÓRIO)				CORPO (OPCIONAL)		
CLA	INS	P1	P2	Lc	DATA FIELD	Le

RESPOSTA APDU		
CORPO (OPCIONAL)	STATUS (OBRIGATÓRIO)	
DATA FIELD	SW1	SW2

FIGURA 3.11 – MENSAGENS APDU DE COMANDO E RESPOSTA /

FONTE: ADAPTADO DE CHEN, 2000.

Existe ainda outro protocolo, chamado TPDU, que pertence à camada de transporte e é utilizado para transporte das mensagens. Os dois tipos de TPDU que estão em uso hoje são o T=0 e T=1. O T=0 faz a transmissão *byte* por *byte* entre o cartão e o computador. Já o T=1 faz a transmissão através de blocos de *bytes* (CHEN, 2000).

Segundo CHEN (2000), toda vez que um cartão é energizado, ele envia uma mensagem inicial para o computador especificando os parâmetros necessários para que a comunicação entre os dois possa ser estabelecida. Essa mensagem chama-se ATR. Ela contém 33 *bytes* e possui diversas informações como a taxa de transmissão, os parâmetros de *hardware* do cartão, o número serial do *chip* e o número de versão.

3.4.7 - *Java Card*

Basicamente a tecnologia *Java Card* torna possível que programas escritos em *Java* rodem em cartões *smart card*. Através dessa tecnologia torna-se possível a construção e distribuição de aplicações *Java* embarcadas em cartões inteligentes de modo seguro e rápido (CHEN, 2000).

Embora se trate de tecnologia *Java* e a forma de codificar, os conceitos da linguagem, a sintaxe e a semântica sejam iguais a qualquer outro programa escrito na linguagem, a tecnologia *Java Card* apresenta algumas peculiaridades, dentre elas as seguintes:

- API: Trata-se de uma biblioteca contendo as informações sobre a tecnologia *Java Card.* Contém várias classes customizadas para tornar possível a programação e desenvolvimento de aplicativos para *smart cards* (CHEN, 2000);

- Arquivo CAP: Devido às limitações do s*mart card* é necessária a conversão das classes *Java* em arquivos CAP. Este arquivo contém a representação binária de uma classe *Java*, com suas informações, os códigos executáveis, as informações de conexão e informações de verificação. O arquivo CAP torna possível que o programa seja carregado e executado dentro do cartão inteligente (CHEN, 2000);

- *Java Card Converter*: Provê os serviços de conversão das classes *Java* em arquivos CAP. Realiza basicamente o papel que uma máquina virtual comum executaria durante o carregamento de uma classe. Atividades como verificar violações da linguagem, inicialização das variáveis, otimização do código e alocação de espaço em memória são realizadas durante a conversão. Após a conversão do arquivo .CLASS de uma classe *Java* é gerado o arquivo CAP (CHEN, 2000);

- JCVM: É a máquina virtual *Java Card.* Diferentemente da máquina virtual *Java* comum, esta é dividida em duas partes. Uma parte está presente dentro do cartão e possui o interpretador do código em *bytes*. A outra parte encontra-se no computador e possui o conversor do código. Juntas as partes implementam a função da máquina virtual que é carregar e executar as classes *Java* (CHEN, 2000);

- JCRE: Responsável por toda a administração de recursos, comunicação de rede e execução do *applet* dentro do cartão (CHEN, 2000).

O uso do *smart card* e sua integração com a identificação biométrica da impressão digital vem proporcionar uma forma mais segura e eficaz de proteger um ambiente de acesso restrito como é o sistema financeiro. Essas tecnologias já são consagradas e comprovadamente eficazes no que se refere ao controle de acesso, autenticação e reconhecimento de usuários. No próximo capítulo é apresentado o desenvolvimento do projeto proposto utilizando as referidas tecnologias.

CAPÍTULO 4 – DESENVOLVIMENTO DO SISTEMA DE AUTENTICAÇÃO

Este capítulo apresenta a proposta de solução e os passos para o desenvolvimento do sistema de autenticação de compras eletrônicas utilizando *smart card* e biometria digital.

4.1 - Proposta para Solução do Problema

Para o desenvolvimento da solução são necessários os seguintes equipamentos:

- um cartão *smart card*;
- uma leitora/gravadora *smart card*;
- um leitor biométrico.

A leitora/gravadora *smart card* visa a gravação dos dados na memória do cartão inteligente e sua posterior leitura. Por sua vez, o leitor biométrico tem por objetivo a captura da impressão digital do usuário.

Uma interface será responsável pela captura e cadastramento da impressão digital juntamente com os demais dados cadastrais do usuário. Como resultado desta captura, obtém-se um código único de caracteres que será utilizado posteriormente para a validação da compra. Todos os dados do cliente, juntamente com o código gerado pela captura da sua impressão digital serão persistidos em banco de dados.

O ATR do cartão, juntamente com o número da conta corrente/cartão, o indicador da modalidade e o número único de identificação do cliente formarão o código que depois de tratado será inserido na memória do cartão para posterior comparação.

Para a realização da compra, o usuário deverá inserir o cartão na leitora, de onde serão retiradas as informações previamente gravadas. A impressão digital do usuário deverá ser novamente capturada, gerando novo código único de caracteres. O sistema irá buscar na base de dados o código da impressão digital pertencente àquela informação de conta

cartão/corrente e aquele usuário obtidos do cartão. Se existir algum usuário correspondente e a impressão digital associada coincidir com a capturada, a primeira validação é realizada, passando-se para a etapa de checagem do cartão e verificação de limite/saldo. Se existir algum usuário correspondente, porém a impressão digital associada não coincidir com a capturada, significa que o cartão não pertence àquela pessoa. Por sua vez, se o resultado da busca não trouxer nenhum registro, significa que não existe nenhum usuário correspondente para aquelas informações.

Para as simulações trabalha-se com a premissa de que todo cartão obrigatoriamente deve ter informações gravadas em sua memória, ou seja, todo cartão deve ter um usuário associado.

4.2 - Pré-Requisitos

O correto funcionamento da aplicação depende dos seguintes pré-requisitos:

- A aplicação deve ser instalada em computador com no mínimo a seguinte configuração: Processador de 1,5 GHz, 1 GB de memória RAM, 20 GB de disco rígido, processador de 32 *bits*, placa de rede habilitada e sistema operacional *Windows XP* ou superior;

- São necessárias no mínimo duas portas USB. Uma para conexão da leitora de *smart card* e outra para a conexão do leitor biométrico;

- Os cartões inteligentes devem suportar a versão 2.2.1 do *Java Card*;

- Instalação e configuração do *Java* no computador;

- Instalação e configuração do banco de dados MySQL;

- Usuário com impressão digital possível de ser capturada pela leitora;

- O usuário deve ser previamente cadastrado e ter apenas um *smart card* vinculado a ele.

4.3 - Tecnologias Utilizadas

4.3.1 - *Softwares*

Os *softwares* utilizados no desenvolvimento e execução da aplicação são:

- GPShell: Ferramenta utilizada para fazer a inserção do arquivo CAP no *smart card;*

- IDE Eclipse: IDE desenvolvida em *Java*, com código aberto para a construção de programas de computador. Usada principalmente para desenvolvimento em *Java* por possuir diversos recursos e ferramentas para a linguagem;

- *Java Card Development Kit* 2.2.1: Disponibiliza toda a API necessária para o desenvolvimento da *Applet* a ser inserida no cartão inteligente. Possibilita também a conversão da classe para o arquivo CAP;

- *Java SE Runtime Environment* 6: Escolhida por ser a versão mais recente disponível no mercado, embora versões anteriores do *Java* (até a versão 3) atendam perfeitamente a aplicação;

- MySQL: O mais popular Sistema de Gerenciamento de Banco de Dados do mercado. Utilizado para persistência das informações;

- SDK Nitgen: Providencia a conversão da impressão digital para o código em caracteres, além disso possui as bibliotecas necessárias para o desenvolvimento de aplicativos utilizando o leitor biométrico fabricado pela Nitgen.

4.3.2 - *Hardwares*

Os equipamentos de *hardware* necessários para o desenvolvimento e execução da aplicação são:

- Leitor biométrico Nitgen Hamster I: Utilizado para captura e verificação da impressão digital;

- Leitora e gravadora de s*mart card* Perto CCID: Responsável pela leitura e gravação dos dados na memória do cartão inteligente;

- *Smart Card* JCOP 2.2.1: Cartão inteligente que suporta a tecnologia *Java Card* 2.2.1.

4.4 – Modelo de Dados

4.4.1 – Diagrama Entidade Relacionamento

Para a persistência dos dados gerados pela aplicação, bem como as posteriores verificações realizadas com esses dados utilizou-se o modelo de dados representado na figura 4.1. São seis tabelas: clientes, conta_cartao, conta_corrente, plastico, reg_conta_corrente, reg_conta_cartao. Essas tabelas são responsáveis respectivamente por armazenar os dados relativos aos usuários da aplicação, os dados das contas cartões dos clientes que optarem por cartão de crédito, os dados das contas correntes dos clientes que optarem por cartão de débito, os dados do plástico, os registros das transações efetuadas com cartões de débito e os registros das transações efetuadas com cartões de crédito.

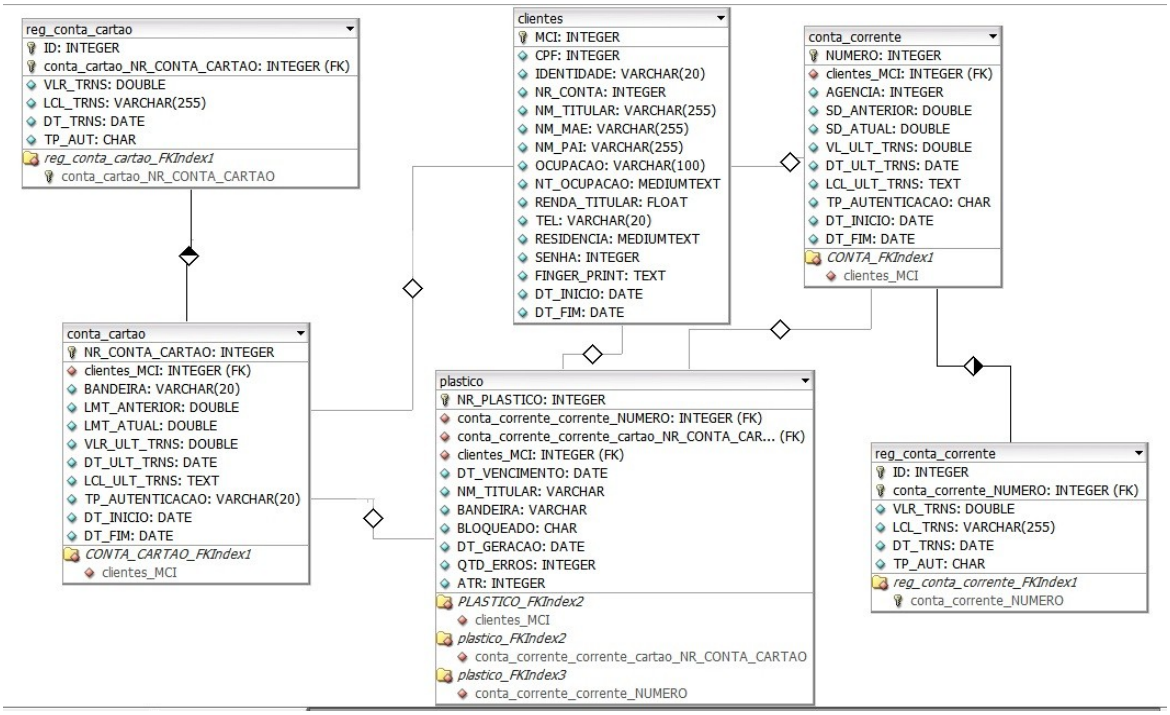

FIGURA 4.1 – MODELO ENTIDADE-RELACIONAMENTO / FONTE: O AUTOR

Como pode-se observar na figura 4.1, cada tabela possui apenas uma chave primária. Na tabela "clientes" a chave primária é o campo "MCI", ou código de identificação do cliente. Na tabela "conta_corrente" a chave primária é o campo "NUMERO", responsável por armazenar o número das contas correntes. Da mesma forma, na tabela "conta_cartão" a chave primária é o campo "NR_CONTA_CARTAO", que guarda o número das contas cartão. Por sua vez, na tabela "plástico", a chave primária é o campo "NR_PLASTICO", que armazena o número dos plásticos gerados pela aplicação após o cadastramento do usuário. Nas tabelas "reg_conta_cartão" e "reg_conta_corrente" o campo "ID" corresponde a chave primária.

Cada cliente pode ter apenas uma operação, conta corrente ou conta cartão e deverá possuir apenas um plástico vinculado a ele. Da mesma forma, cada plástico está vinculado a apenas uma operação. As tabelas "reg_conta_cartão" e "reg_conta_corrente" podem apresentar diversos registros de uma determinada conta cartão ou conta corrente, apresentando um relacionamento N:1 com as tabelas "conta_corrente" e "conta_cartão".

4.4.2 – Diagrama de Fluxo de Dados

O sistema está dividido em dois módulos: módulo cadastramento, responsável pelo cadastramento da impressão digital e das informações cadastrais do cliente e vinculação deste a um cartão; e módulo terminal POS, responsável pela simulação do ambiente de compras e seus diversos fluxos. A figura 4.2 representa o diagrama de fluxo do módulo cadastramento e os passos para execução de todo o processo de cadastramento.

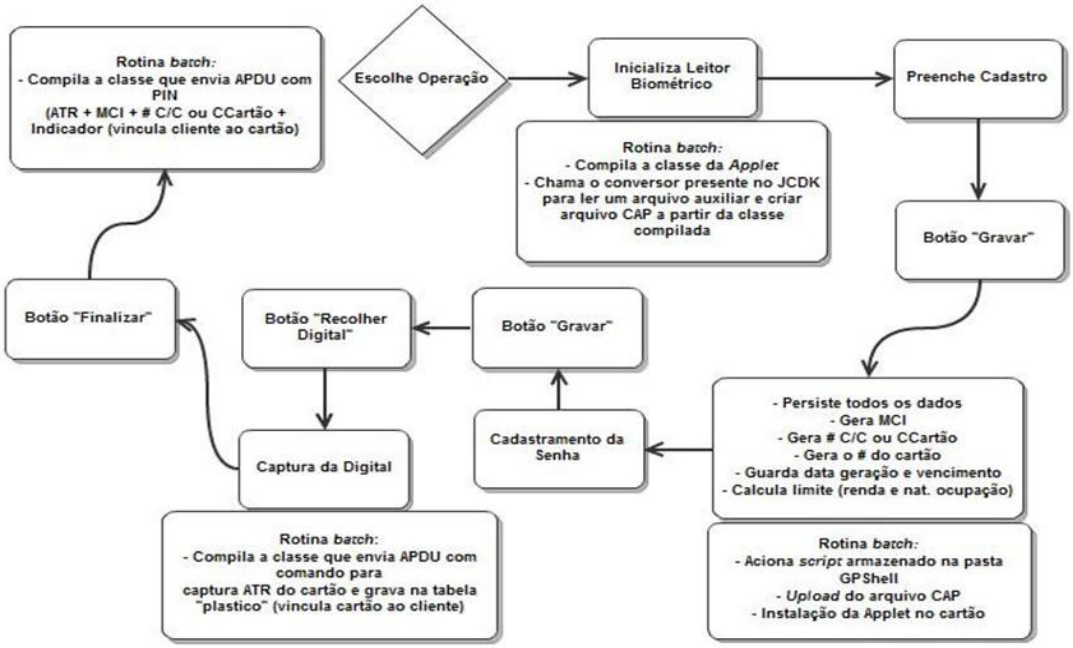

FIGURA 4.2 - FLUXO MÓDULO CADASTRAMENTO / FONTE: O AUTOR

Da mesma forma, na figura 4.3 observa-se o diagrama de fluxo do módulo terminal POS e os passos necessários para a efetivação de uma compra no ambiente simulado.

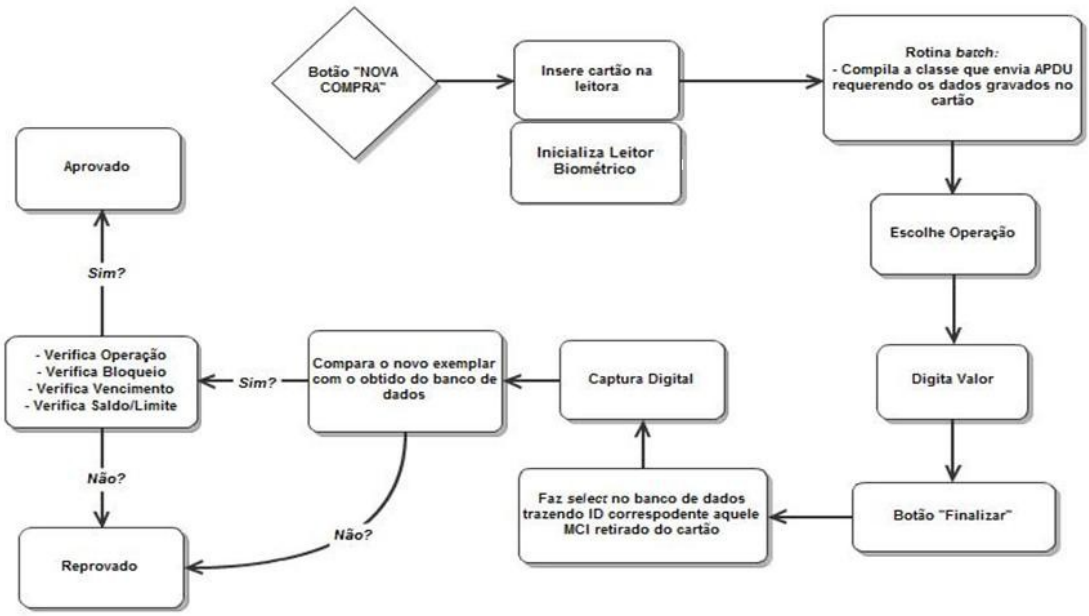

FIGURA 4.3 - FLUXO MÓDULO TERMINAL POS / FONTE: O AUTOR

4.5 – Desenvolvimento da Aplicação

4.5.1 – Cadastramento do Usuário

O processo de cadastramento tem início quando o usuário escolhe a operação que deseja selecionar, como ilustra a figura 4.4. Cada cliente deverá escolher entre uma conta-corrente ou um cartão de crédito. Assim que se seleciona a operação desejada, os campos do cadastro podem ser preenchidos e neste momento também ocorre a conexão do leitor biométrico e a execução de mais dois processos. Um deles compila a classe *Java* da *applet* que será inserida no cartão, o outro é responsável pela geração do arquivo CAP da *applet.*

O leitor biométrico utilizado no projeto é de fabricação da empresa Nitgen. O equipamento vem acompanhado de um kit de desenvolvimento e de *software* responsável pela captura e conversão da impressão digital. Este *software* proprietário foi utilizado para aquisição e tratamento das imagens geradas pela captura da impressão digital dos usuários, uma vez que o desenvolvimento de tal programa não faz parte do escopo do projeto.

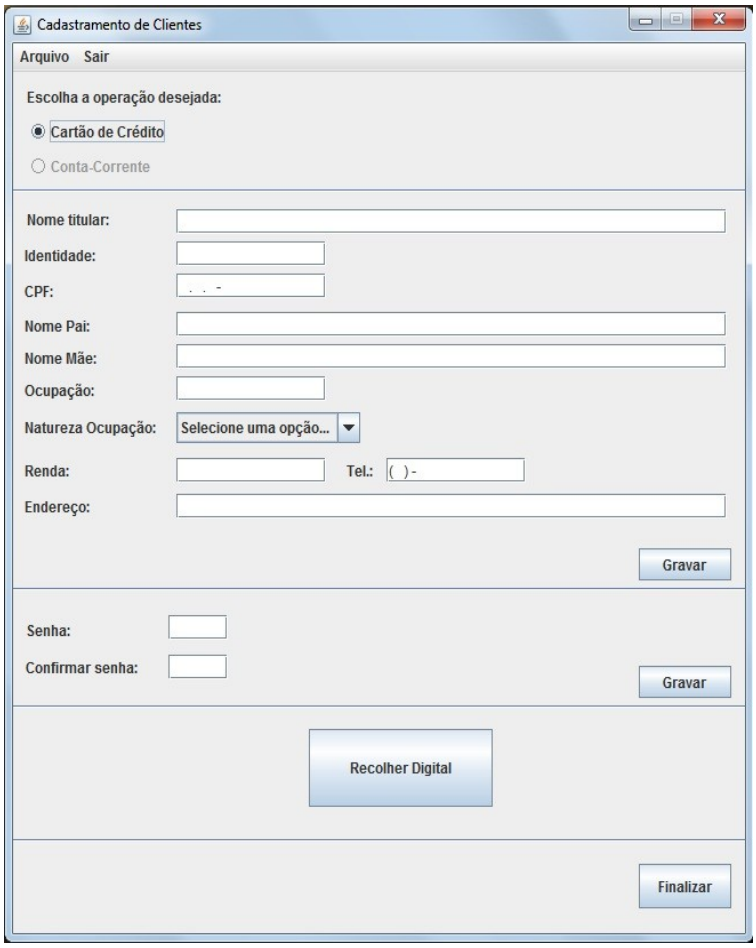

FIGURA 4.4 - TELA DE CADASTRO / FONTE : O AUTOR

Após a escolha da operação torna-se possível cadastrar o usuário. O cadastro é relativamente complexo, uma vez que visa se aproximar o máximo possível do encontrado em um ambiente bancário real. São ao todo dez campos a serem preenchidos: nome do titular, identidade, CPF, nome do pai, nome da mãe, ocupação, natureza da ocupação, renda, telefone e endereço como apresentado na figura 4.5. Após o preenchimento de todos os campos clica-se no botão “Gravar”. Este botão é responsável por acionar os métodos responsáveis pelas seguintes tarefas:

- Gerar o código de identificação do cliente;
- Gerar o número da conta corrente ou conta cartão;
- Gerar o número do cartão do cliente;

- Inserir na base de dados as datas de início de relacionamento, de geração do cartão e de seu respectivo vencimento;
- Calcular o limite da conta corrente/cartão do cliente. O limite calculado dependerá da renda e da natureza da ocupação do cliente e será inserido na base de dados para posterior utilização durante a fase de compras. Esta regra de negócio foi implementada também visando se adequar o máximo possível à realidade bancária. Transferências, depósitos, pagamentos ou qualquer outra operação bancária que provoque alterações no saldo/limite do cliente não fazem parte do escopo do projeto.

Serão providenciados também neste momento a instalação da *applet* no *smart card* e a inserção do arquivo CAP na memória do cartão através da ferramenta GPShell.

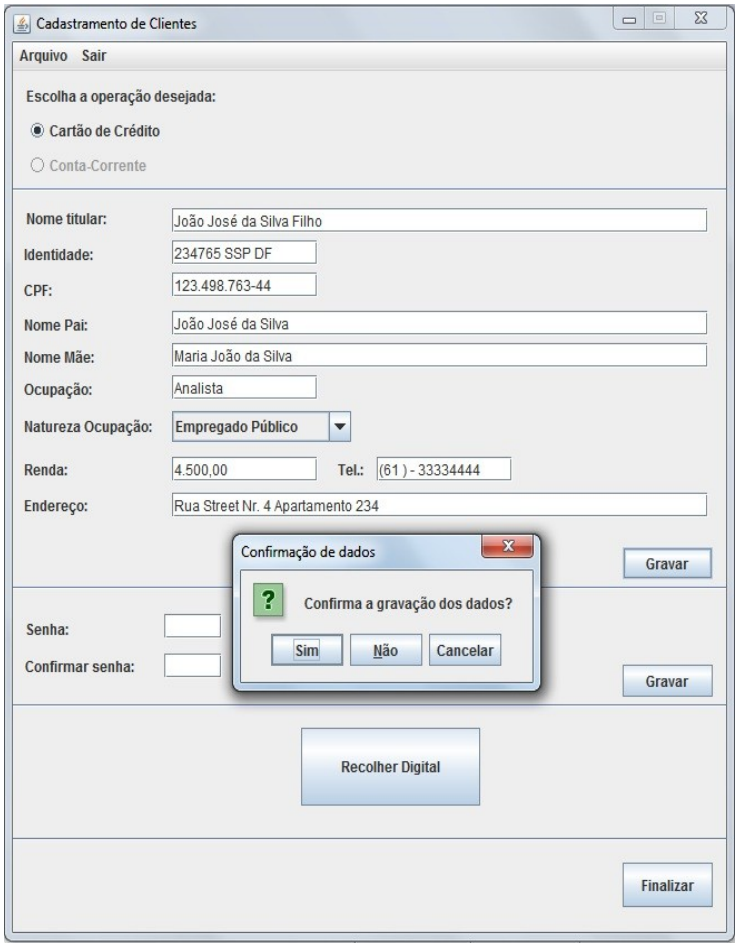

FIGURA 4.5 - TELA DE CADASTRO 2 / FONTE: O AUTOR

Logo após esta etapa será solicitado que o cliente cadastre uma senha de seis dígitos (figura 4.6). Esta senha será usada no caso de impossibilidade de autenticar a compra através da captura da impressão digital do cliente devido a alguma doença ou acidente. Essa possibilidade não será contemplada durante a fase de testes, uma vez que foge o escopo do projeto.

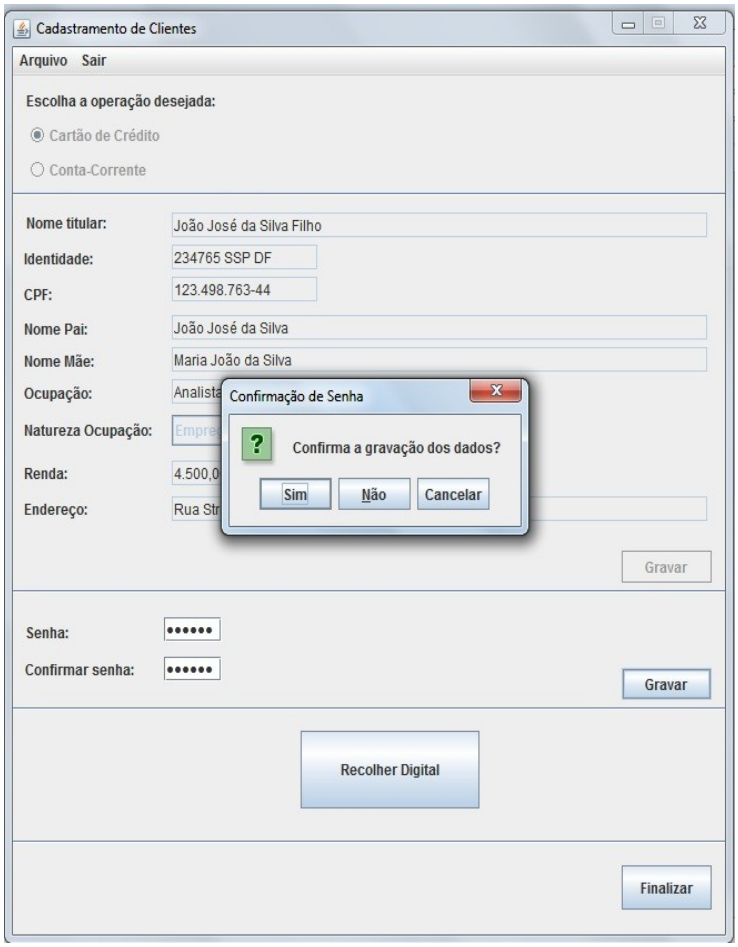

FIGURA 4.6 - TELA DE CADASTRO 3 / FONTE: O AUTOR

Isto feito, o usuário deverá clicar no botão “Recolher Digital”. Este botão acionará o *software* proprietário da Nitgen, responsável pela captura e conversão da impressão digital em código de caracteres (figura 4.7). Este código será posteriormente persistido em banco de dados.

FIGURA 4.7 - CAPTURA DA DIGITAL / FONTE: O AUTOR

Ao final desse processo, outra rotina é executada, desta vez pela JCVM, gerando o número ATR do cartão que será gravado na tabela “plástico”.

Por fim, o ATR do cartão, o código de identificação do cliente (MCI), o número da conta corrente/cartão e o indicador da modalidade são concatenados gerando o PIN a ser inserido no cartão. Uma outra rotina é responsável por compilar e executar a classe *Java* que envia a mensagem APDU que insere o PIN na memória do cartão. O PIN passa por um *hash code* antes de ser inserido.

Ao final deste processo o usuário estará cadastrado e devidamente vinculado a um cartão.

4.5.2 – Autenticação do Usuário

Para a autenticação do cliente e verificação se este está autorizado a completar a transação, outro aplicativo foi desenvolvido. Este aplicativo simula a aparência e a operacionalização de um terminal POS (figura 4.8).

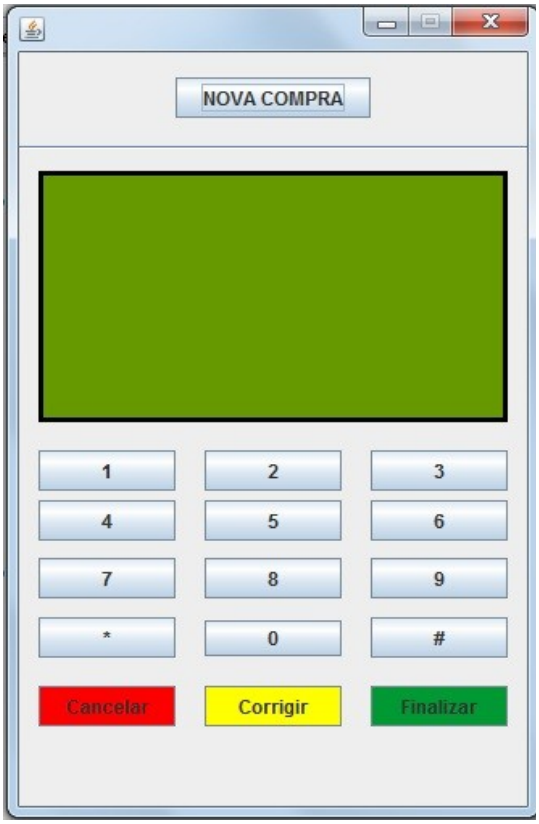

FIGURA 4.8 - SIMULAÇÃO TERMINAL POS / FONTE: O AUTOR

Ao se clicar no botão “NOVA COMPRA”, o usuário é solicitado a inserir o cartão *smart card* na leitora. Após inserido, o cliente escolhe a operação desejada, débito ou crédito, e digita o valor da transação.

Passada esta etapa, clica-se no botão “Finalizar”. Como resultado desta ação, novamente o *software* proprietário da Nitgen é acionado, recolhendo novo exemplar da impressão digital e fazendo sua conversão para código de caracteres (figura 4.9). O sistema

então procura na base de dados o cliente correspondente àquele código identificador e conta corrente/cartão obtidos do *smart card*. Se existir um registro correspondente, então ele compara se a impressão digital armazenada para aquele cliente é compatível com o novo exemplar recolhido no momento da compra.

FIGURA 4.9 - VERIFICAÇÃO DA DIGITAL / FONTE: O AUTOR

4.5.3 – Autorização da Compra

A completa autorização da compra pelo sistema depende de três etapas principais. Primeiro o sistema tenta encontrar no banco de dados cliente correspondente para as informações de conta corrente/cartão e código do cliente retiradas do cartão. Após esta etapa, verifica se a impressão digital do cliente encontrado é compatível com o novo exemplar recolhido no momento da compra. Passada esta segunda fase, agora o sistema verifica se o

cartão não se encontra bloqueado e se existe saldo/limite disponível para realização da compra.

As etapas acima descritas simulam de maneira simplificada o comportamento de uma compra real e tem por objetivo garantir a integridade da transação. Se alguma das etapas não for autorizada a compra é recusada. Todas as compras efetivadas são registradas em banco de dados para posterior auditoria e verificação.

4.6 – Estimativa de Custos

QUADRO 4.1 - CUSTOS DO PROJETO

Equipamentos	**Valor**
Leitor Biométrico Nitgen	R$ 235,90
Leitor/Gravador *Smart Card* Perto	R$ 69,90
Cartão *Smart Card*	R$ 15,00
Softwares	R$ 0,00
Total =	R$ 320,80

FONTE: O AUTOR

CAPÍTULO 5 – APLICAÇÃO DO SISTEMA E RESULTADOS DOS TESTES

Este capítulo apresenta a aplicação do sistema em vários testes realizados. Os testes visam simular o cadastramento e as posteriores tentativas de compra de um cliente fictício.

5.1 – Casos de Teste

5.1.1 – Cadastro de cliente

Cadastro e vinculação ao cartão do cliente fictício José da Silva Filho

Resultado Esperado: “Cadastro Finalizado com sucesso”.

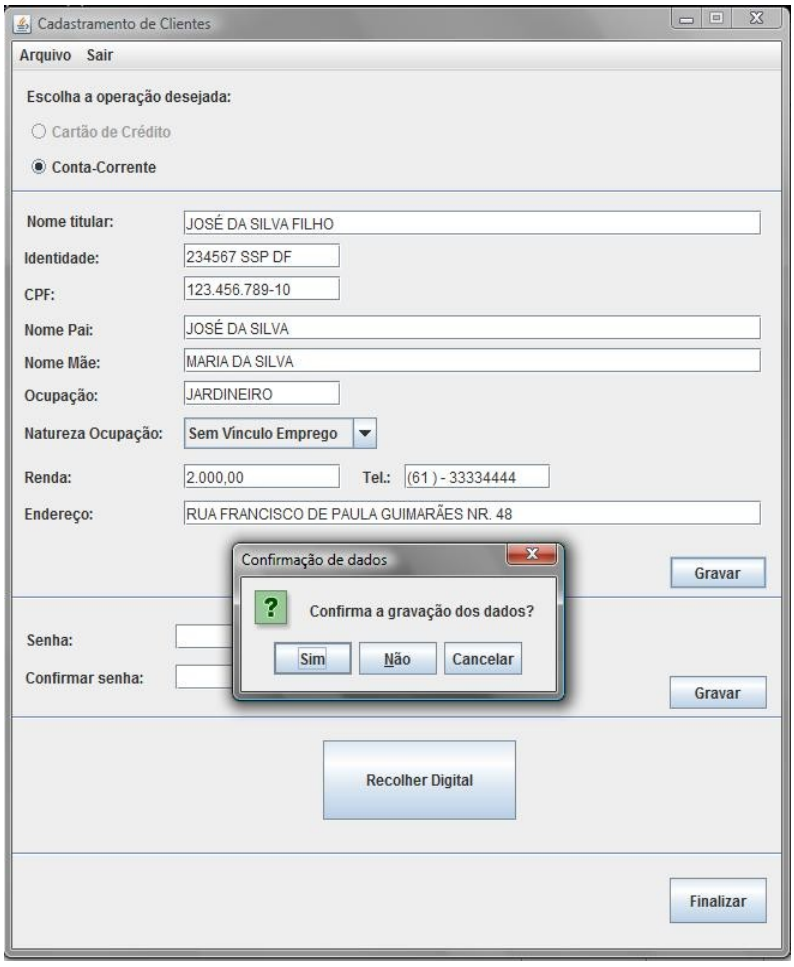

FIGURA 5.1 - CADASTRO DE CLIENTE - TELA 1 / FONTE: O AUTOR

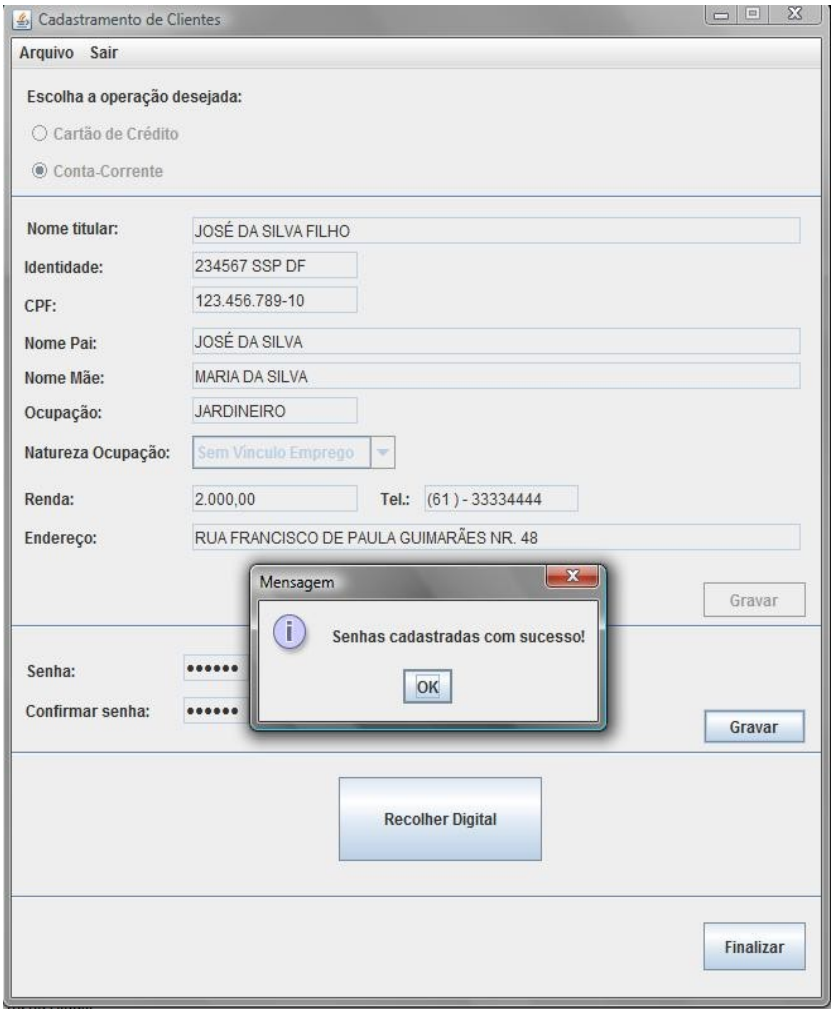

FIGURA 5.2 - CADASTRO DE CLIENTE - TELA 2 / FONTE: O AUTOR

As figuras 5.1 e 5.2 ilustram respectivamente as etapas de escolha da operação e posterior preenchimento do cadastro e a etapa de cadastramento da senha de seis dígitos do cliente fictício José da Silva Filho. No teste realizado o cadastro foi efetuado com sucesso, bem como a geração dos dados do cliente (número MCI, número da conta corrente, número do plástico e cálculo de limite) e a persistência desses dados em banco de dados.

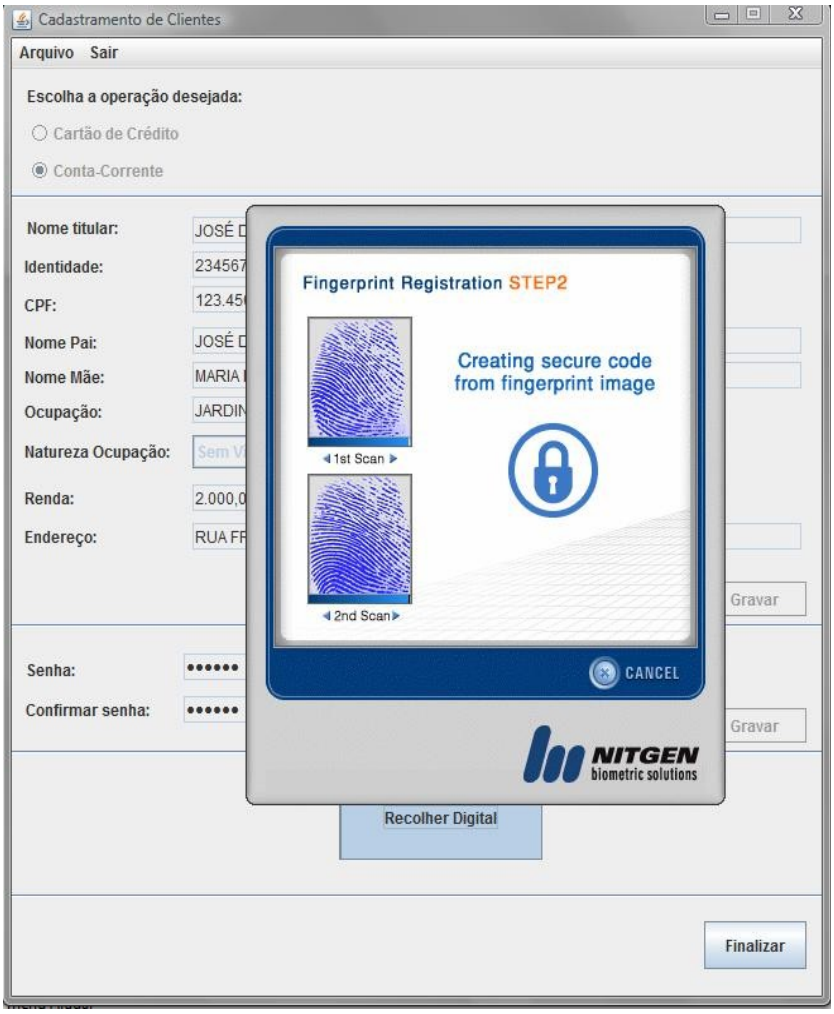

FIGURA 5.3 - CADASTRO DE CLIENTE - TELA 3 / FONTE: O AUTOR

Na figura 5.3 é apresentada a etapa de cadastramento da impressão digital do cliente fictício José da Silva Filho. O cadastramento ocorre através do acionamento do *software* proprietário da empresa Nitgen. No teste realizado o cadastramento foi feito com sucesso, bem como a persistência em banco de dados do *template* gerado pela captura.

O *template* gerado pela captura da impressão digital apresenta-se na forma de uma *string* ou código de caracteres como ilustra a figura 5.4. Esse *template* apresenta-se diferente mesmo para capturas de exemplares iguais da mesma pessoa, o que explica-se pela alteração de fatores como luminosidade e rotação do dedo em cada captura. Todavia, o sistema biométrico é capaz de identificar os padrões necessários para a autenticação, ressaltando umas das principais características desses sistemas, a flexibilidade.

•AQAAABQAAAAEAwAAAQASAAMAXAAAAAAAAAAMAALMy2OyQrMbNby9QCw*jM*ZPmV6SCsOU8RxFBkpaybe4i*
aguWUUzmTv0yHRD9/6gK8IGTb3Nog1ERjZc/NCuLvse3X5Kp7ZzwzUFxZYa*bWPblggaVWRKmartTBqL*N1cjoUFD
MtrTQwShaTPFQIpM2RBR2p478MLbC*9xunUjYQ9mN48/zMwIEKqX*yYRI7xePuvEbl0I1smp3Fn9tYTV4sp9FaKvCLO
/b7iYfsfOIPAW35i4NsHVl/qGlbMo7Y0JgtGz5LRjXkted4QPL/TXWMIv2U7O7UQwLCBRwqtUtBDV3sXMHyHzyILml/qM
9DGTyUykwI8yP4GXmzdoMHvazr8ENe*BzE0fhwO8tPzm*XXiOI*oVm52O/FV4G2GJ3Il4Csh5w7dfq3pZez*ZRzOVzO
W5qhGw8tBs8deDmLGYIsCcBDvcpFcc7WI7mUjwgOFTumqtLPy*wZuyADneHxeE8GmFxsasxGAOqlga/iiv6yrcsPee6
nVXmSLDvdwby3bxSC7h/vNI*AdiEpAevnCY*tI8lI5DGvNdNjITCN7qVrnGPDKqlKn0UvFCEzCpi/6h6ldKPFcK46/xIbUupj
h/gB1Fsh5/kLOGL*yD/OKvFbWGnKcNnWuhEMtWJR/ZHryVYucdxGubbQCCkbc5vrG5da25UQx1OWkuG6dJQOp5zy
WxIUxLSO6M7bISHSzR8xVJd95RFsuZ7YLA4I76JDIi5RVn2FiO*h44yEgYMN3hWMAu*BV5IFt0qmdGTzhi6ViV80MWl
Q0nfm8EhBi5BIFLxMbNDPvxD4EunOiGXL/*nRTtDZmB/e/oIGCEcW2NczC7b0qW2fSyPqHAy/QcH3raJdTGVLQfi5mS
A*CJmyXyDzJUuhJzEHl3d795*H2LwU29Ou5rAHQAdfA2xsB9IfsiNUD64Btra5d/clp8fnKrPtnaxI5SsK8VdKTmCnQpkIlg
QUts6mtOMhaRYjqISXuNIemHeITG*9LFxkZILvONj9yJYyndeO3j2C/5YopdQg

FIGURA 5.4 - CÓDIGO GERADO PELA CAPTURA DA ID / FONTE: O AUTOR

Como apresentado na figura 5.5, o processo de cadastramento do cliente fictício foi finalizado com sucesso. Nessa etapa também concluíu-se como esperado o processo de extração do ATR do cartão para gravação em banco de dados, bem como a vinculação do cliente ao cartão inteligente através da gravação do PIN na memória deste.

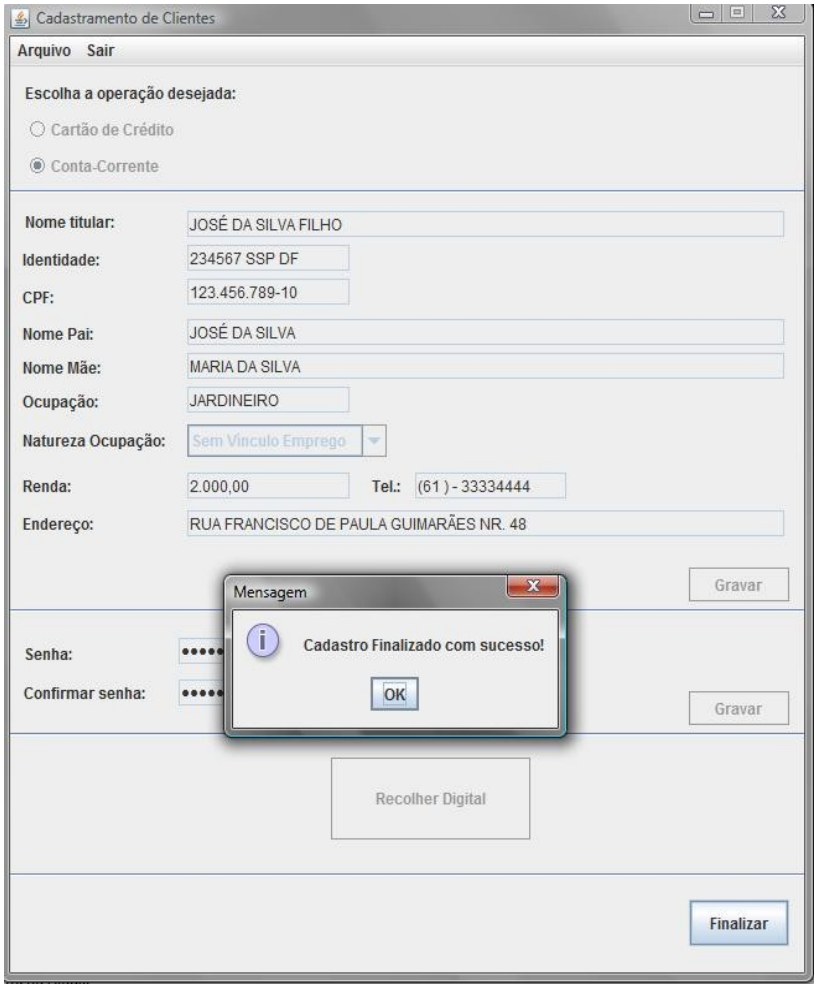

FIGURA 5.5 - CADASTRO DE CLIENTE - TELA 4 / FONTE: O AUTOR

5.1.2 – Compra com cartão com saldo suficiente em conta-corrente

Tentativa de compra do cliente fictício José da Silva Filho na função débito no valor de R$ 300,00. Saldo em conta corrente de R$ 600,00.

Resultado Esperado: Compra “Aprovada”.

FIGURA 5.6 - TERMINAL POS - TELA 1 /
FONTE: O AUTOR

A figura 5.6 ilustra a tela inicial do aplicativo de simulação do terminal POS. Nesta etapa o operador inicialmente aciona o botão “NOVA COMPRA”. O sistema então solicita que o cartão seja inserido na leitora *smart card* para a extração dos dados armazenados em sua memória. No teste realizado a extração dos dados armazenados no cartão e sua posterior utilização pela aplicação *host* ocorreu como esperado.

FIGURA 5.7 - TERMINAL POS - TELA 2 /
FONTE: O AUTOR

A figura 5.7 ilustra a etapa de solicitação do valor da compra simulada. Essa etapa ocorre logo após a escolha da operação, débito ou crédito, por parte do operador.

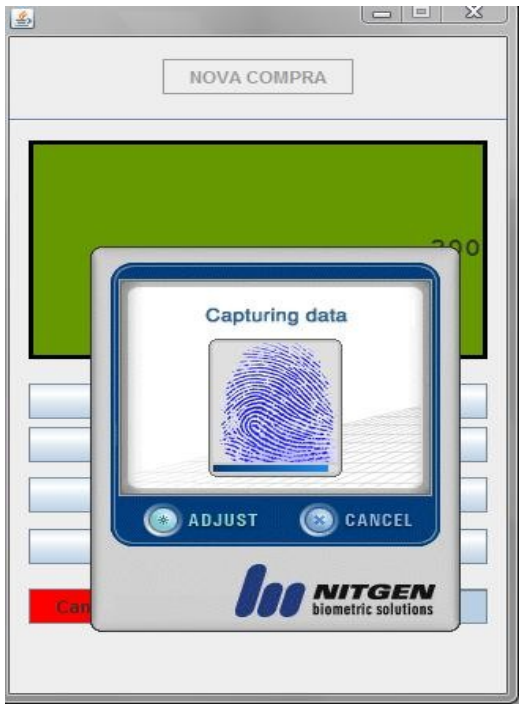

FIGURA 5.8 - TERMINAL POS - TELA 3 /
FONTE: O AUTOR

Logo após a digitação do valor, o operador pressiona o botão “Finalizar”, acionando novamente o *software* proprietário da Nitgen. Novo exemplar da impressão digital foi então capturado como apresentado na figura 5.8.

Como esperado, após a realização das etapas acima descritas e a comparação do novo exemplar recolhido com o exemplar armazenado em banco de dados, a transação foi aprovada como ilustra a figura 5.9.

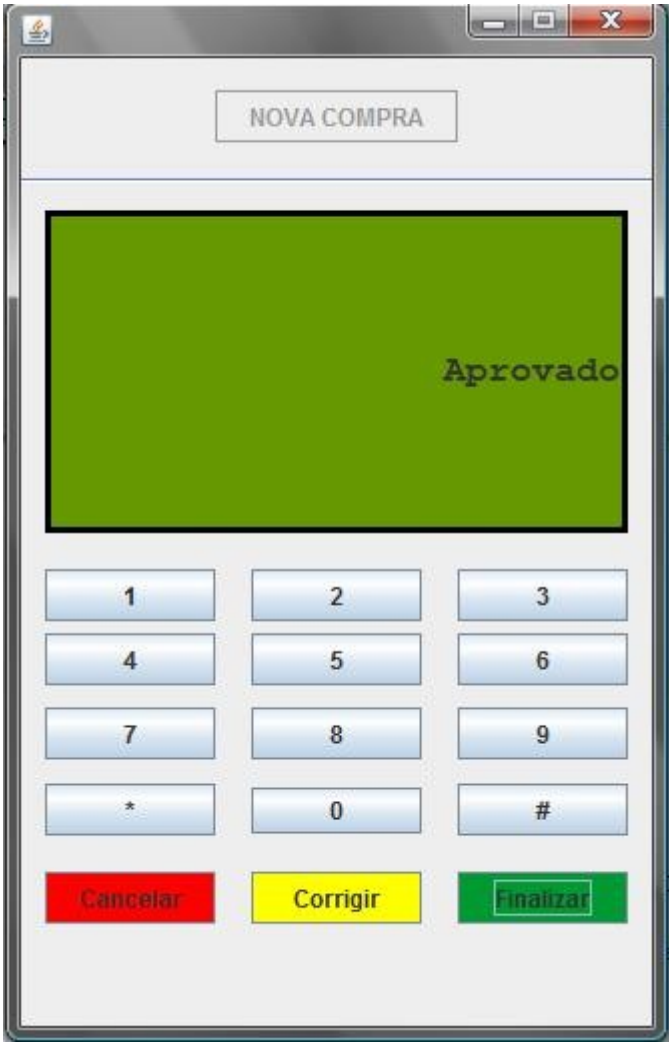

FIGURA 5.9 - TRANSAÇÃO APROVADA / FONTE: O AUTOR

5.1.3 – Compra com cartão com saldo insuficiente em conta-corrente

Tentativa de compra do cliente fictício José da Silva Filho na função débito no valor de R$ 700,00. Saldo em conta corrente de R$ 300,00.

Resultado Esperado: Recusa por “Saldo Insuficiente”.

Os passos apresentados nas figuras 5.6, 5.7 e 5.8 são também aqui previamente executados.

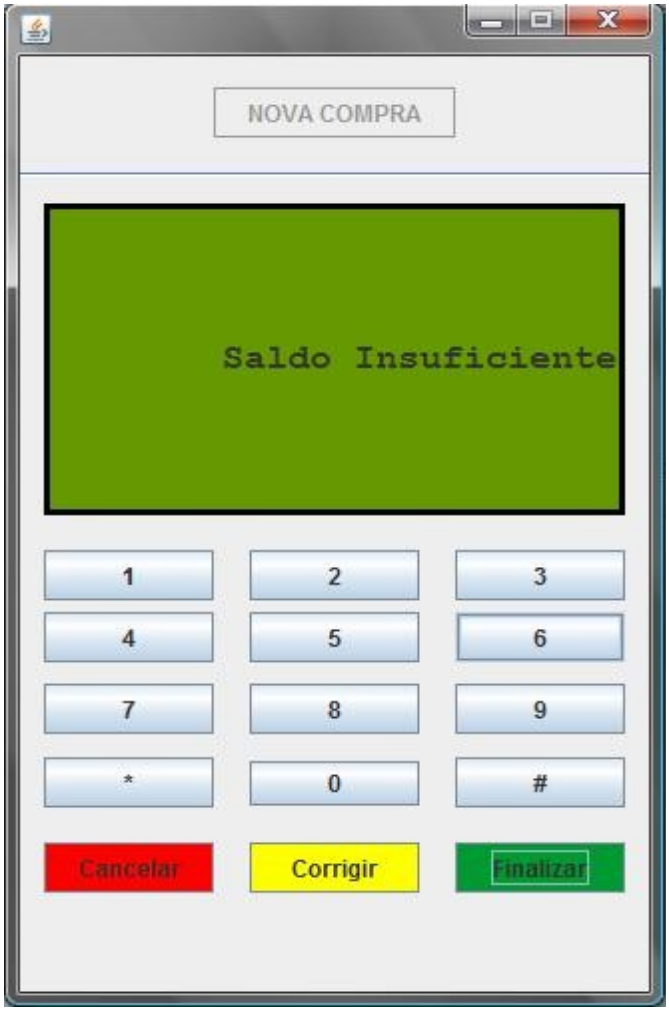

FIGURA 5.10 - TRANSAÇÃO REPROVADA - SALDO INSUFICIENTE / FONTE: O AUTOR

5.1.4 – Compra com cartão com saldo suficiente em conta-corrente e impressão digital inválida

Tentativa de compra de outra pessoa utilizando o cartão do cliente fictício José da Silva Filho na função débito no valor de R$ 200,00. Saldo em conta corrente de R$ 300,00.

Resultado Esperado: Recusa por “Digital Inválida”.

Os passos apresentados nas figuras 5.6, 5.7 e 5.8 são também aqui previamente executados.

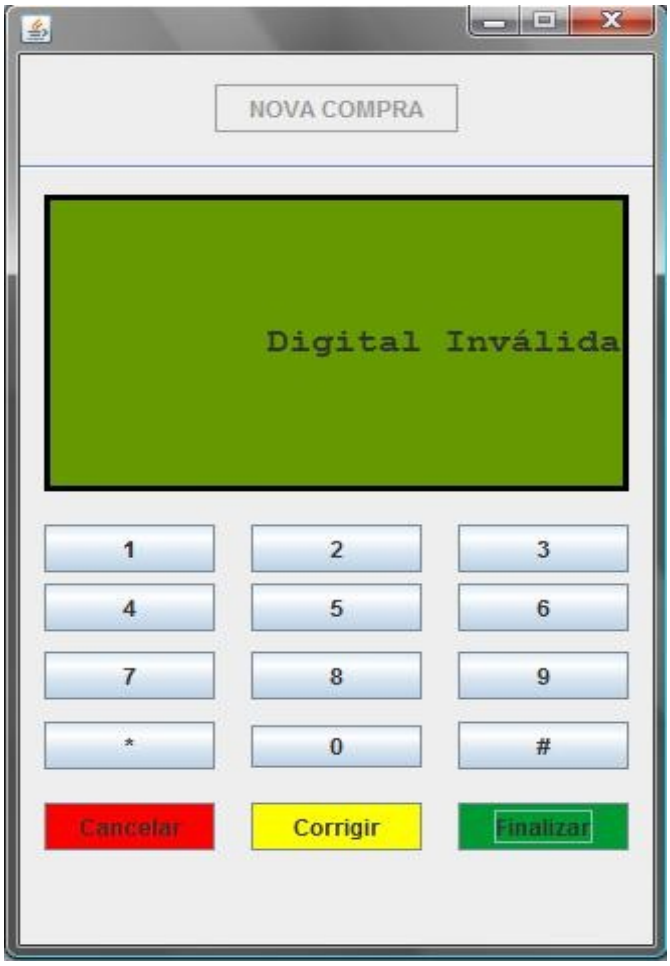

FIGURA 5.11 - TRANSAÇÃO REPROVADA - DIGITAL INVÁLIDA / FONTE: O AUTOR

5.1.5 – Compra com cartão escolhendo a função incorreta

Tentativa de compra do cliente fictício José da Silva Filho na função crédito no valor de R$ 120,00. O cliente possui apenas conta corrente.

Resultado Esperado: Recusa por “Operação Incorreta”.

Os passos apresentados nas figuras 5.6, 5.7 e 5.8 são também aqui previamente executados.

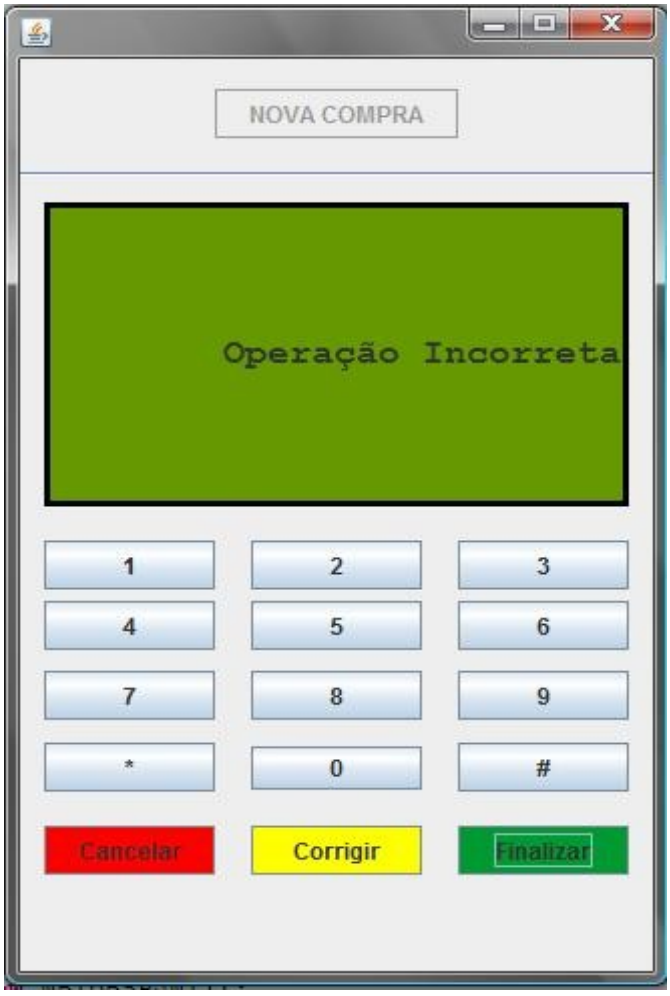

FIGURA 5.12 - TRANSAÇÃO REPROVADA – OPERAÇÃO INCORRETA / FONTE: O AUTOR

5.1.6 – Compra com cartão bloqueado

Tentativa de compra do cliente fictício José da Silva Filho na função débito no valor de R$ 200,00. Saldo em conta corrente de R$ 300,00. O cartão está bloqueado por três ou mais tentativas de acesso inválidas.

Resultado Esperado: Recusa por “Cartão Bloqueado”.

Os passos apresentados nas figuras 5.6, 5.7 e 5.8 são também aqui previamente executados.

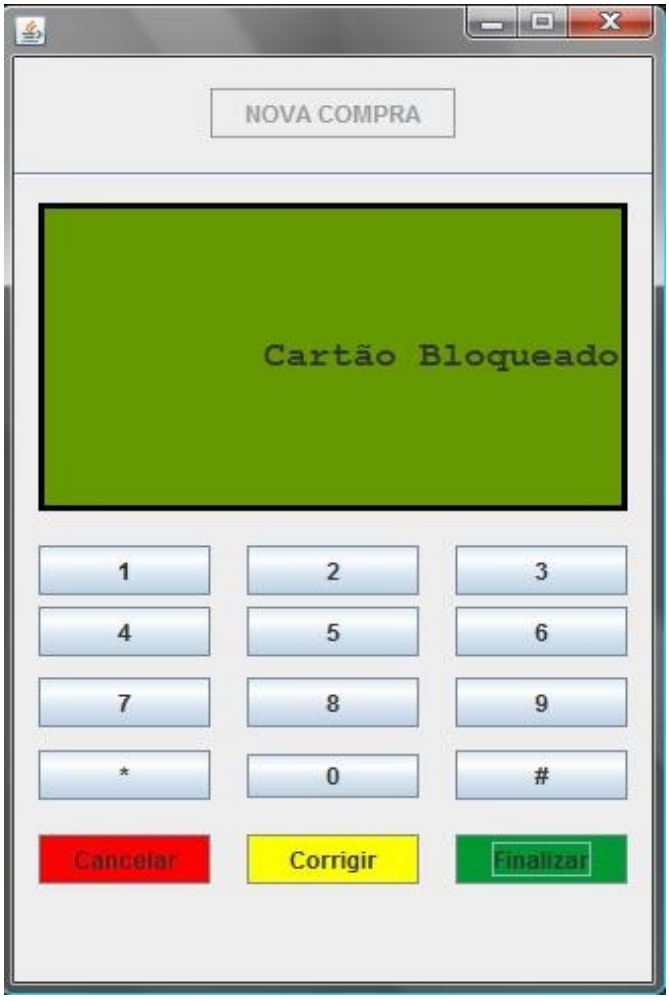

FIGURA 5.13 - TRANSAÇÃO REPROVADA - CARTÃO BLOQUEADO / FONTE: O AUTOR

5.1.7 – Quadro de Resultados dos Testes

QUADRO 5.1 - RESULTADOS DOS TESTES

Caso de Teste	Resultado
Cadastro de cliente	Satisfatório
Compra com cartão com saldo suficiente em conta-corrente	Satisfatório
Compra com cartão com saldo insuficiente em conta-corrente	Satisfatório
Compra com cartão com saldo suficiente em conta-corrente e impressão digital inválida	Satisfatório
Compra com cartão escolhendo a função incorreta	Satisfatório
Compra com cartão bloqueado	Satisfatório

FONTE: O AUTOR

5.2 – Análise dos Resultados

5.2.1 – Pontos Positivos

O sucesso na realização do cadastro, na captura da impressão digital do usuário e na vinculação deste ao cartão *smart card* demonstram a eficiência do projeto. Da mesma maneira, todo o processo de autenticação da compra simulada, reconhecimento do usuário, extração de dados do cartão e as consequentes contabilizações foram executadas como esperado e proposto.

5.2.2 – Dificuldades e Desvantagens

Durante a execução do projeto algumas dificuldades foram encontradas, dentre elas:

- Poucos materiais e referências sobre tecnologia S*mart Card* e *Java Card.* Apesar de não serem muito novas, poucos profissionais da área se dedicam ao estudo dessas tecnologias, o que tornou-se um obstáculo para o desenvolvimento do projeto*;*

- A memória limitada do cartão não suportou a gravação do código de caracteres gerado pela captura da impressão digital. Desta forma, o uso de banco de dados para gravação do mesmo tornou-se necessário;

- O *smart card* utilizado tem suporte para a versão 2.2.1 do *Java Card.* Essa versão ainda não apresenta a API de biometria, o que foi implementado a partir da versão 2.2.2. Além disso, os *plugins* para desenvolvimento em *Java Card* instalados na IDE Eclipse só são compatíveis com a versão 2.2.2 do JCDK;

- O driver do leitor biométrico não é compatível com processadores de 64 *bits*. Algumas horas de análise foram despendidas até a descoberta desta informação. Posteriormente o computador utilizado no projeto teve de ser substituído.

CAPÍTULO 6 – CONCLUSÃO

6.1 - Síntese Conclusiva

Todo recurso, principalmente financeiro deve estar cercado do máximo de segurança e confiabilidade possível. As metodologias de segurança e controle de acesso baseadas na capacidade do homem de memorizar ou manter algo consigo têm se mostrado muitas vezes ineficazes.

O homem é suscetível a falhas, podendo esquecer ou até mesmo emprestar suas senhas ou códigos de acesso, desconhecendo o perigo que corre ao praticar tal ato. Dessa forma, agregaram-se nesse projeto duas metodologias de segurança comprovadamente eficazes, buscando somar os seus benefícios e aplicá-las ao ambiente financeiro, de forma que se pudesse garantir o máximo de segurança e confiabilidade possível.

Como resultado do projeto obteve-se um protótipo que integra tecnologia *smart card* e biometria digital, baseado nos conceitos “do que você tem” e “do que você é”. Foram realizados 6 (seis) testes simulando situações diversas, todos eles apresentando resultados satisfatórios.

De maneira geral, os objetivos gerais e específicos foram cumpridos, uma vez que a junção das duas tecnologias propostas, bem como a sua integração com o sistema foi realizada com sucesso.

A utilização conjunta dessas duas tecnologias e sua aplicação ao ambiente financeiro traz uma nova proposta de acesso às compras no crédito e débito, aproveitando o baixo índice de falha das biometrias e a segurança contra clonagem obtida com os cartões inteligentes.

6.2 - Sugestões para Trabalhos Futuros

Levando-se em conta as dificuldades encontradas durante a realização do projeto e o grande potencial de aplicação das tecnologias utilizadas, sugere-se a realização dos seguintes projetos:

- Construção de solução semelhante utilizando outro tipo de autenticação biométrica (íris, reconhecimento facial, etc.);

- Construção de solução semelhante utilizando versões superiores da tecnologia *Java Card*, de modo que a necessidade do banco de dados seja eliminada;

- Utilização das mesmas tecnologias para construção de outra aplicação (acesso a áreas restritas, controle de entrada e saída, controle de ponto, etc.).

REFERÊNCIAS BIBLIOGRÁFICAS

ABECS, Associação Brasileira de Empresas de Cartão de Crédito e Serviços. **Monitor ABECS.** 2010**.** Disponível em: <http://www.abecs.org.br>. Acesso em: 04 de Outubro de 2010.

ABNT, Associação Brasileira de Normas Técnicas. **Cartão Plástico com Circuito Integrado – Smart Cards Cartões com Memória,** [20--]. Cartilha ABNT – CB 21, SC 21:01, FEBRABAN, SIMPRO, CT012, CT013, Primeira Revisão.

ANDRADE, Paulo Gustavo Sampaio. **Direito das obrigações e contratos, Cartão de crédito**. 2002. Artigo. Disponível em: <http://jus2.uol.com.br/doutrina/texto.asp?id=621>. Acesso em: 20 de agosto de 2010.

ARAÚJO, Paulo Gabriel Ribacionka Góes . **Sistema de Controle de Acesso via S*mart Card* com Autenticação Biométrica da Impressão Digital.** Monografia de Graduação do Curso de Engenharia de Computação. Brasília: UniCEUB, 2º semestre de 2010.

ASA, Associação dos Supermercados de Alagoas. **LEI Nº 4.132, DE 2 DE MAIO DE 2008. Autoria do Projeto: Deputado Raad Massouh,** 2008. Disponível em <www.asa-al.com.br/downloads/**lei4132**.pdf >. Acesso em: 13 de abril de 2011.

ASHBOURN, Julian. **The distinction between authentication and identification**, [S.l.], 2000. Disponível em: <http://homepage.ntlworld.com/avanti/authenticate.htm>. Acesso em: 25 de outubro de 2010.

BB, Banco do Brasil. **LIC – Livro de Instruções Codificadas, Fraudes contra cartões.** Brasília, 2010.

CBA, **Consultores Biométricos Associados**. Rio de Janeiro. [20--]. Disponível em <http://www.consultoresbiometricos.com.br/index.php>. Acesso em 05 de outubro de 2010.

CHEN**, Ziqun. Java Card Technology for Smart Cards**. Prentice Hall, 2000.

CONECTADOS. **Amigos do caminho.** Disponível em <http://www.conectados.blogger.com.br >. Acesso em: 05 de outubro de 2010.

COSTA, Silvia Maria Farani. **Classificação e reconhecimento de impressões digitais**. 2000. 30 f. Dissertação (Mestrado em Engenharia Elétrica) – Escola Politécnica da Universidade de São Paulo, São Paulo. Disponível em: <http://sim.lme.usp.br/ publicacoes/exames/pdf/QualiSi.pdf>. Acesso em: 02 de outubro de 2010.

DUARTE, Ângelo. TEDESCO, Cláudia. COUTO, Daniel Lucena. **Desenvolvimento de algoritmos para análise de imagens de impressões digitais rotacionadas utilizando grafos.** 2004. 15 f. Disponível em <http://www.frb.br/ciente/Impressa/Info/I. 1.Duarte,A.Desenvolvimento%20de%20Algoritmos....pdf>. Acesso em 02 de outubro de / 2010.

FEBRABAN, Federação Brasileira de Bancos. **O setor bancário em números. Relatório Junho 2010. 2010**. 18 f. Disponível em: <http://www.febraban.org.br/p5a_52gt34++5cv8_4466+ff145afbb52ffrtg33fe36455li5411pp+e/sitefebraban/Setor_Banc%E1rio_N%FAmeros_Junho_2010%20%282%29.pdf>. Acesso em: 12 de agosto de 2010.

_____, Federação Brasileira de Bancos. **Ciab Febraban 2009. " Bancarização " Coletiva - O Setor Bancário em Números.** 2009. 17 f. Disponível em <http://www.febraban.org.br/p5a_52gt34++5cv8_4466+ff145afbb52ffrtg33fe36455li5411pp+e/sitefebraban/Apresenta%E7%E3o%20-%20O%20Setor%20banc%E1rio%20em%20N%FAmeros.ppt.site.pdf>. Acesso em: 12 de agosto 2010

FERREIRA, Aurélio Buarque de Holanda. **Dicionário Aurélio eletrônico século XXI**. Direção Geral de Carlos Augusto Lacerda. Rio de Janeiro: Nova Fronteira, 1999. 1 CD-ROM. Produzido por Lexikon Informática.

FRAUDES, Monitor das Fraudes. **Fraudes no Comércio e C.D.C.,** 2009. Disponível em: <http://www.fraudes.org/>. Acesso em: 04 de outubro de 2010

GUMZ, Rafael Araújo. **Protótipo de um sistema de identificação de minúcias em impressões digitais utilizando redes neurais artificiais feedforward multicamada.** 2002. Trabalho de Conclusão de Curso – Universidade Regional de Blumenau – Santa Catarina.

HONG, Lin. **Automatic personal identification using fingerprints**. 1998. 242 f. Dissertação (Doutorado em Filosofia) – Departamento de Ciências da Computação, Michigan State University, Ann Arbor. Disponível em: <http://www.cse.msu.edu/publications/tech/TR/MSUCPS-98-24.ps.gz>. Acesso em: 14 novembro de 2010.

INÁCIO, Sandra Regina da Luz. **Prevenção de fraudes contra os cartões de crédito**. Disponível em <http://artigos.netsaber.com.br/resumo_artigo_28694/artigo_sobre_prevencao_de_fraudes_contra_os_cartoes_de_credito> Acesso em: 30 de maio de 2011.

_____, Sandra Regina da Luz. **As fraudes em cartão de crédito**. Disponível em <http://www.soartigos.com/artigo/1041/As-Fraudes-em-Cartoes-de-Credito> Acesso em: 30 de maio de 2011.

KEHDY, Carlos. **Elementos de criminalística**. 3.ed. São Paulo: Sugestões Literárias, 1968.

LEÒNCIO, HENRIQUE C. M. **O Uso de Certificados Digitais ICP Brasil, Padrão A3, Como Tecnologia de Acesso a Conta-Corrente em Canal de Auto-Atendimento Internet**. Brasília. 2006.

LIMA, Márcia. **Bradesco cadastra 1 mi para biometria**. 2009. Artigo. Disponível em: <http://www.baguete.com.br/noticias/geral/31/08/2009/bradesco-cadastra-1-mi-para-biometria>. Acesso em 04 de outubro de 2010.

MAZI, Renan Corio, DAL PINO JÚNIOR, Arnaldo. **Identificação biométrica através da impressão digital usando redes neurais artificiais**. 2009. 09 f. Instituto Militar de Engenharia – ITA.

OBELHEIRO, Rafael R.; COSTA, Luciano R.; FRAGA, Joni S. [20--] **Introdução à Biometria**. Departamento de Automação e Sistemas - Universidade Federal de Santa Catarina.

PANKANTI, Sharath. **Biometrics: promising frontiers for emerging identification market**, Howthorne, 2000. 16 f. Disponível em: <http://www.cse.msu.edu/publications/tech/TR/MSUCSE-00-2.ps. gz>. Acesso em: 02 de outubro de 2010.

PINHEIRO, José Maurício**. Biometria nos Sistemas Computacionais - Você é a Senha**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2008.

REVISTA DIGITAL, Uma publicação do Instituto Nacional de Tecnologia da Informação – ITI. **Certificação Digital - Certificação digital e biometria trazem mais segurança para empresas, governos e cidadãos.** Revista. Ano 1 – nº 2 – 2º semestre 2009.

SONSUN**. Kit de Desenvolvimento JAVACARD.** Disponível em <http://www.sonsun.com.br/sonsun_JCOP.php> Acesso em: 10 de março de 2011.

SOUSA, Giovanni Ferreira. **Controle de ponto utilizando a tecnologia Java Card.** Monografia de Graduação do Curso de Engenharia de Computação. Brasília: UniCEUB, 2º Semestre de 2009.

SOUZA, Marcelo Barbosa. **Controle de Acesso: Conceitos, Tecnologia e Benefícios.** São Paulo: Editora Sicurezza, 2010.

SUN, Java. **Java Card Development Kit 2.2.1**. Disponível em <http://java.sun.com/javacard/devkit/> Acesso em: 10 de maio de 2011.

TAVARES JÚNIOR, Gilberto da Silva. **A papiloscopia nos locais de crime**. São Paulo: Ícone, 1991.

WAYMAN, J. L. (1999a). **Error rate equations for the general biometric system**. *IEEE Robotics & Automation Magazine*, 6(1):35–48.

APÊNDICES

Transcreve-se abaixo código da interface do aplicativo de cadastramento:

```
package projetofinal;

import java.awt.Color;
import java.sql.SQLException;
import java.text.DecimalFormat;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.JOptionPane;

import br.com.vinicius.Cliente;
import br.com.vinicius.ContaCartao;
import br.com.vinicius.ContaCorrente;
import br.com.vinicius.Plastico;
import br.com.vinicius.action.GeraDados;
import br.com.vinicius.action.GeraLimites;
import br.com.vinicius.action.GravaBancoDados;
import br.com.vinicius.smartcard.CompilaArquivoBat;

import com.nitgen.SDK.BSP.NBioBSPJNI;

public class Cadastro extends javax.swing.JFrame {

        private static final long serialVersionUID = 1448567054490342326L;

        private NBioBSPJNI.IndexSearch objIndexSearch;
        private NBioBSPJNI bsp;
        DecimalFormat df = new DecimalFormat("#,###.00");

        public Cadastro() {

      initComponents();

   }

   Cliente cliente = new Cliente();

   private void initComponents() {

      labelNomeTitular = new javax.swing.JLabel();
      labelIdentidade = new javax.swing.JLabel();
      labelCPF = new javax.swing.JLabel();
      labelNomePai = new javax.swing.JLabel();
```

```
labelNomeMae = new javax.swing.JLabel();
labelOcupacao = new javax.swing.JLabel();
labelNaturezaOcupacao = new javax.swing.JLabel();
labelRenda = new javax.swing.JLabel();
labelEndereco = new javax.swing.JLabel();
campoNomeTitular = new javax.swing.JTextField();
campoIdentidade = new javax.swing.JTextField();
campoNomePai = new javax.swing.JTextField();
campoNomeMae = new javax.swing.JTextField();
campoOcupacao = new javax.swing.JTextField();
campoEndereco = new javax.swing.JTextField();
caixaNaturezaOcupacao = new javax.swing.JComboBox();
botaoGravarCadatro = new javax.swing.JButton();
labelSenha = new javax.swing.JLabel();
labelConfirmarSenha = new javax.swing.JLabel();
campoSenha = new javax.swing.JPasswordField();
campoConfirmarSenha = new javax.swing.JPasswordField();
botaoGravarSenha = new javax.swing.JButton();
botaoRecolherDigital = new javax.swing.JButton();
labelOpcaoOperacao = new javax.swing.JLabel();
selecaoCartaoCredito = new javax.swing.JRadioButton();
selecaoContaCorrente = new javax.swing.JRadioButton();
botaoFinalizar = new javax.swing.JButton();
labelAlerta = new javax.swing.JLabel();
campoRenda = new javax.swing.JFormattedTextField();
campoCPF = new javax.swing.JFormattedTextField();
labelTelefone = new javax.swing.JLabel();
campoTelefone = new javax.swing.JFormattedTextField();
separadorSenha = new javax.swing.JSeparator();
separadorOperacao = new javax.swing.JSeparator();
separadorDigital = new javax.swing.JSeparator();
separadorCadastro = new javax.swing.JSeparator();
labelAlerta1 = new javax.swing.JLabel();
barraMenu = new javax.swing.JMenuBar();
botaoMenuArquivo = new javax.swing.JMenu();
subMenuArquivoSair = new javax.swing.JRadioButtonMenuItem();
botaoMenuSair = new javax.swing.JMenu();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Cadastramento de Clientes");
setResizable(false);

labelNomeTitular.setText("Nome titular:");

labelIdentidade.setText("Identidade:");

labelCPF.setText("CPF:");

labelNomePai.setText("Nome Pai:");
```

```
        labelNomeMae.setText("Nome Mãe:");

        labelOcupacao.setText("Ocupação:");

        labelNaturezaOcupacao.setText("Natureza Ocupação:");

        labelRenda.setText("Renda:");

        labelEndereco.setText("Endereço:");

        caixaNaturezaOcupacao.setModel(new javax.swing.DefaultComboBoxModel(new
String[] { "Selecione uma opção...", "Empresário", "Empregado Público", "Empregao
Privado", "Sem Vínculo Emprego" }));

        botaoGravarCadatro.setText("Gravar");
        botaoGravarCadatro.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoGravarCadatroActionPerformed(evt);
            }
        });

        labelSenha.setText("Senha:");

        labelConfirmarSenha.setText("Confirmar senha:");

        botaoGravarSenha.setText("Gravar");
        botaoGravarSenha.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoGravarSenhaActionPerformed(evt);
            }
        });

        botaoRecolherDigital.setText("Recolher Digital");
        botaoRecolherDigital.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoRecolherDigitalActionPerformed(evt);
            }
        });

        labelOpcaoOperacao.setText("Escolha a operação desejada:");

        selecaoCartaoCredito.setText("Cartão de Crédito");
        selecaoCartaoCredito.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                selecaoCartaoCreditoActionPerformed(evt);
            }
        });
```

```
        selecaoContaCorrente.setText("Conta-Corrente");
        selecaoContaCorrente.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                selecaoContaCorrenteActionPerformed(evt);
            }
        });

        botaoFinalizar.setText("Finalizar");
        botaoFinalizar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoFinalizarActionPerformed(evt);
            }
        });

        campoRenda.setFormatterFactory(new javax.swing.text.DefaultFormatterFactory(new
javax.swing.text.NumberFormatter(new java.text.DecimalFormat("#,###.00"))));

        try {
            campoCPF.setFormatterFactory(new javax.swing.text.DefaultFormatterFactory(new
javax.swing.text.MaskFormatter("###.###.###-##")));
        } catch (java.text.ParseException ex) {
            ex.printStackTrace();
        }
        campoCPF.setCursor(new java.awt.Cursor(java.awt.Cursor.TEXT_CURSOR));

        labelTelefone.setText("Tel.:");

        try {
            campoTelefone.setFormatterFactory(new
javax.swing.text.DefaultFormatterFactory(new javax.swing.text.MaskFormatter("(## ) -
########")));
        } catch (java.text.ParseException ex) {
            ex.printStackTrace();
        }

        botaoMenuArquivo.setText("Arquivo");
        botaoMenuArquivo.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoMenuArquivoActionPerformed(evt);
            }
        });

        subMenuArquivoSair.setSelected(true);
        subMenuArquivoSair.setText("Sair");
        subMenuArquivoSair.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                subMenuArquivoSairActionPerformed(evt);
            }
        });
```

```
        botaoMenuArquivo.add(subMenuArquivoSair);

        barraMenu.add(botaoMenuArquivo);

        botaoMenuSair.setText("Sair");
        botaoMenuSair.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                botaoMenuSairMouseClicked(evt);
            }
        });
        barraMenu.add(botaoMenuSair);

        setJMenuBar(barraMenu);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(10, 10, 10)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                            .addComponent(labelRenda)
                            .addComponent(labelEndereco)
                            .addComponent(labelNaturezaOcupacao)
                            .addComponent(labelIdentidade)
                            .addComponent(labelCPF)
                            .addComponent(labelNomePai)
                            .addComponent(labelNomeMae)
                            .addComponent(labelOcupacao)))
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(labelNomeTitular)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addComponent(campoNomePai, javax.swing.GroupLayout.DEFAULT_SIZE,
478, Short.MAX_VALUE)
                    .addComponent(campoNomeMae, javax.swing.GroupLayout.DEFAULT_SIZE,
478, Short.MAX_VALUE)
                    .addComponent(campoOcupacao,
javax.swing.GroupLayout.PREFERRED_SIZE, 129,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(caixaNaturezaOcupacao,
javax.swing.GroupLayout.PREFERRED_SIZE, 161,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addGroup(layout.createSequentialGroup()
                    .addComponent(campoRenda, javax.swing.GroupLayout.PREFERRED_SIZE,
131, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELAT
ED)
                    .addComponent(labelTelefone)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(campoTelefone,
javax.swing.GroupLayout.PREFERRED_SIZE, 122,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addComponent(campoEndereco, javax.swing.GroupLayout.DEFAULT_SIZE,
478, Short.MAX_VALUE)
                .addComponent(campoNomeTitular,
javax.swing.GroupLayout.DEFAULT_SIZE, 478, Short.MAX_VALUE)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRA
ILING, false)
                    .addComponent(campoIdentidade,
javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(campoCPF, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.PREFERRED_SIZE, 125,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGap(16, 16, 16))
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(labelAlerta)
            .addContainerGap(603, Short.MAX_VALUE))
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addComponent(selecaoContaCorrente)
                .addComponent(selecaoCartaoCredito)
                .addComponent(labelOpcaoOperacao))
            .addContainerGap(459, Short.MAX_VALUE))
        .addComponent(separadorOperacao,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 613, Short.MAX_VALUE)
        .addComponent(separadorDigital, javax.swing.GroupLayout.DEFAULT_SIZE, 613,
Short.MAX_VALUE)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addComponent(labelConfirmarSenha)
                .addComponent(labelSenha))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
```

```
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(campoConfirmarSenha,
javax.swing.GroupLayout.DEFAULT_SIZE, 52, Short.MAX_VALUE))
                .addGroup(layout.createSequentialGroup()
                    .addGap(27, 27, 27)
                    .addComponent(campoSenha, javax.swing.GroupLayout.DEFAULT_SIZE, 52,
Short.MAX_VALUE)))
            .addGap(360, 360, 360)
            .addComponent(botaoGravarSenha)
            .addContainerGap())
        .addComponent(separadorSenha, javax.swing.GroupLayout.DEFAULT_SIZE, 613,
Short.MAX_VALUE)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addContainerGap(231, Short.MAX_VALUE)
            .addComponent(botaoRecolherDigital,
javax.swing.GroupLayout.PREFERRED_SIZE, 161,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(221, 221, 221))
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(labelAlerta1)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 522,
Short.MAX_VALUE)
            .addComponent(botaoGravarCadatro)
            .addContainerGap())
        .addComponent(separadorCadastro, javax.swing.GroupLayout.DEFAULT_SIZE, 613,
Short.MAX_VALUE)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addContainerGap(532, Short.MAX_VALUE)
            .addComponent(botaoFinalizar)
            .addContainerGap())
    );

    layout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {campoConfirmarSenha, campoSenha});

    layout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {campoCPF, campoOcupacao, campoRenda});

    layout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {botaoFinalizar, botaoGravarCadatro, botaoGravarSenha});

    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
```

```
            .addContainerGap()
            .addComponent(labelOpcaoOperacao)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(selecaoCartaoCredito)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(selecaoContaCorrente)
            .addGap(7, 7, 7)
            .addComponent(separadorOperacao,
javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addComponent(campoNomeTitular,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(labelNomeTitular))
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addGroup(layout.createSequentialGroup()
                    .addGap(9, 9, 9)
                    .addComponent(labelIdentidade))
                .addGroup(layout.createSequentialGroup()
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(campoIdentidade,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addGroup(layout.createSequentialGroup()
                    .addGap(6, 6, 6)
                    .addComponent(campoCPF, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(layout.createSequentialGroup()
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELAT
ED)
                    .addComponent(labelCPF)))
            .addGap(9, 9, 9)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addGroup(layout.createSequentialGroup()
                    .addGap(3, 3, 3)
                    .addComponent(labelNomePai))
                .addComponent(campoNomePai, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(6, 6, 6)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addGroup(layout.createSequentialGroup()
```

```
                .addGap(3, 3, 3)
                .addComponent(labelNomeMae))
            .addComponent(campoNomeMae,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
            .addGroup(layout.createSequentialGroup()
                .addGap(9, 9, 9)
                .addComponent(labelOcupacao))
            .addGroup(layout.createSequentialGroup()
                .addGap(6, 6, 6)
                .addComponent(campoOcupacao,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGap(9, 9, 9)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
            .addComponent(caixaNaturezaOcupacao,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(labelNaturezaOcupacao))
        .addGap(12, 12, 12)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
            .addComponent(labelRenda)
            .addComponent(campoRenda, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(campoTelefone, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(labelTelefone))
        .addGap(9, 9, 9)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
            .addComponent(campoEndereco, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(labelEndereco))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 24,
Short.MAX_VALUE)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAIL
ING)
            .addComponent(labelAlerta1)
            .addComponent(botaoGravarCadatro))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(separadorCadastro, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(20, 20, 20)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAIL
ING)
```

```
                .addGroup(layout.createSequentialGroup()
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.T
RAILING)
                        .addComponent(campoSenha,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(labelSenha))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELAT
ED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.B
ASELINE)
                        .addComponent(campoConfirmarSenha,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(labelConfirmarSenha))
                    .addGap(21, 21, 21))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(botaoGravarSenha)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
)
            .addComponent(separadorSenha, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(labelAlerta)
            .addGap(11, 11, 11)
            .addComponent(botaoRecolherDigital,
javax.swing.GroupLayout.PREFERRED_SIZE, 63,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(26, 26, 26)
            .addComponent(separadorDigital, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
            .addComponent(botaoFinalizar, javax.swing.GroupLayout.PREFERRED_SIZE, 36,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(20, 20, 20))
        );

        pack();
    }// </editor-fold>

    private void botaoRecolherDigitalActionPerformed(java.awt.event.ActionEvent evt) {

        //variável temporária usada durante o acionamento do software na Nitgen
        NBioBSPJNI.FIR_HANDLE hSavedFIR;
        hSavedFIR = bsp.new FIR_HANDLE();

        //abre dispositivo
        bsp.OpenDevice();
        //método que aciona o software da Nitgen
```

```
        bsp.Enroll(hSavedFIR, null);
        //fecha dispositivo
        bsp.CloseDevice();

        //variável de armazenamento da impressão digital
        NBioBSPJNI.FIR_TEXTENCODE textEncodeFIR;
        textEncodeFIR = bsp.new FIR_TEXTENCODE();

        //tratamento antes da persistência
        bsp.GetTextFIRFromHandle(hSavedFIR, textEncodeFIR);

        NBioBSPJNI.INPUT_FIR inputFIR = bsp.new INPUT_FIR();
        inputFIR.SetTextFIR(textEncodeFIR);

        //popula o bean
        cliente.setFingerPrintCliente(textEncodeFIR.TextFIR);

        //rotina que recupera numero ATR do cartao
                CompilaArquivoBat cab = new CompilaArquivoBat();
        cab.carregaArquivoBat("CompilaExecuta_RecuperarATR");

        //grava em banco de dados
        try {
                        GravaBancoDados.gravaImpressaoDigital(cliente);
                        JOptionPane.showMessageDialog(campoNomePai, "Digital cadastrada
com sucesso!");
                } catch (Exception e) {
                        e.printStackTrace();
                        JOptionPane.showMessageDialog(campoNomePai, "Falha ao cadastrar
digital!");
                }

      botaoRecolherDigital.setEnabled(false);

}

   private void botaoMenuArquivoActionPerformed(java.awt.event.ActionEvent evt) {


   }

   private void botaoGravarCadatroActionPerformed(java.awt.event.ActionEvent evt) {
                if(selecaoCartaoCredito.isSelected() || selecaoContaCorrente.isSelected()){

             if(campoNomeTitular.getText() == null ||
("").equals(campoNomeTitular.getText())){

                labelAlerta1.setVisible(true);
                labelAlerta1.setForeground(Color.red);
```

```
    labelAlerta1.setText("PREENCHA O NOME DO TITULAR!");
    return;
}

if(campoIdentidade.getText() == null || ("").equals(campoIdentidade.getText())){

    labelAlerta1.setVisible(true);
    labelAlerta1.setForeground(Color.red);
    labelAlerta1.setText("PREENCHA A IDENTIDADE DO TITULAR!");
    return;
}

if(campoCPF.getText() == null || ("   .   .   -  ").equals(campoCPF.getText())){

    labelAlerta1.setVisible(true);
    labelAlerta1.setForeground(Color.red);
    labelAlerta1.setText("PREENCHA O CPF DO TITULAR!");
    return;
}

if(campoNomePai.getText() == null || ("").equals(campoNomePai.getText())){

    labelAlerta1.setVisible(true);
    labelAlerta1.setForeground(Color.red);
    labelAlerta1.setText("PREENCHA O NOME DO PAI DO TITULAR!");
    return;
}
if(campoNomeMae.getText() == null || ("").equals(campoNomeMae.getText())){

    labelAlerta1.setVisible(true);
    labelAlerta1.setForeground(Color.red);
    labelAlerta1.setText("PREENCHA O NOME DA MÃE DO TITULAR!");
    return;
}

if(campoOcupacao.getText() == null || ("").equals(campoOcupacao.getText())){

    labelAlerta1.setVisible(true);
    labelAlerta1.setForeground(Color.red);
    labelAlerta1.setText("PREENCHA A OCUPAÇÃO DO TITULAR!");
    return;
}
if(caixaNaturezaOcupacao.getSelectedItem() == "Selecione uma opção..."){

    labelAlerta1.setVisible(true);
    labelAlerta1.setForeground(Color.red);
    labelAlerta1.setText("SELECIONE A NATUREZA DA OCUPAÇÃO!");
    return;
}
```

```
            if(campoRenda.getText() == null || ("").equals(campoRenda.getText())){

               labelAlerta1.setVisible(true);
               labelAlerta1.setForeground(Color.red);
               labelAlerta1.setText("PREENCHA A RENDA DO TITULAR!");
               return;
            }

            if(campoTelefone.getText() == null || ("(   ) -
").equals(campoTelefone.getText())){

               labelAlerta1.setVisible(true);
               labelAlerta1.setForeground(Color.red);
               labelAlerta1.setText("PREENCHA O TELEFONE DO TITULAR!");
               return;

            }
            if(campoEndereco.getText() == null || ("").equals(campoEndereco.getText())){

               labelAlerta1.setVisible(true);
               labelAlerta1.setForeground(Color.red);
               labelAlerta1.setText("PREENCHA O ENDEREÇO DO TITULAR!");
               return;
            }

            int opcao = JOptionPane.showConfirmDialog(campoNomePai, "Confirma a
gravação dos dados?",
                  "Confirmação de dados", JOptionPane.YES_NO_CANCEL_OPTION);

            //rotina que instala Applet na memoria do cartao
            CompilaArquivoBat cab = new CompilaArquivoBat();
               cab.carregaArquivoBat("Instala_AppletSmartCard");

            boolean flag;

            flag = opcao == JOptionPane.YES_OPTION;

               if( flag ){

                  labelAlerta1.setVisible(false);
                  campoNomeTitular.setEditable(false);
                  campoIdentidade.setEditable(false);
                  campoCPF.setEditable(false);
                  campoNomePai.setEditable(false);
                  campoNomeMae.setEditable(false);
                  campoOcupacao.setEditable(false);
                  caixaNaturezaOcupacao.setEnabled(false);
                  campoRenda.setEditable(false);
                  campoEndereco.setEditable(false);
```

```
                campoTelefone.setEditable(false);
                botaoGravarCadatro.setEnabled(false);
                selecaoCartaoCredito.setEnabled(false);
                selecaoContaCorrente.setEnabled(false);


                try {

                    cliente.setMciCliente(GeraDados.geraMCICliente());
                    cliente.setCpfCliente(campoCPF.getText());
                    cliente.setIdCliente(campoIdentidade.getText());
                    cliente.setNrContaCliente(GeraDados.geraNrContaCorrente());
                    cliente.setNomeCliente(campoNomeTitular.getText());
                    cliente.setNomePaiCliente(campoNomePai.getText());
                    cliente.setNomeMaeCliente(campoNomeMae.getText());
                    cliente.setOcupacaoCliente(campoOcupacao.getText());

cliente.setNatOcupacaoCliente(caixaNaturezaOcupacao.getSelectedItem().toString());
                    cliente.setTelefoneCliente(campoTelefone.getText());
                    cliente.setRendaCliente(campoRenda.getText().replace(".", ""));
                    cliente.setResidenciaCliente(campoEndereco.getText());

cliente.setDtInicioRelCliente(GeraDados.geraDataInicioRelacionamento());

                    JOptionPane.showMessageDialog(campoNomePai, "Cadastro efetuado
com sucesso!");

                    if(selecaoCartaoCredito.isSelected()){

                        ContaCartao contaCartao = new ContaCartao();
                        Plastico plastico = new Plastico();

                        /* DADOS DA CONTA CARTAO*/
                        contaCartao.setLimiteAtual(GeraLimites.limiteCartao(cliente));
                        contaCartao.setMciCliente(cliente.getMciCliente());
                        contaCartao.setNrContaCartao(GeraDados.geraNrContaCartao());

        contaCartao.setDtInicioContaCartao(GeraDados.geraDataInicioRelacionamento());

                        /* DADOS DO PLASTICO*/
                        plastico.setMciCliente(cliente.getMciCliente());
                        plastico.setNrPlastico(GeraDados.geraNrPlastico());
                        plastico.setNrContaCorrente(0);
                        plastico.setNrContaCartao(contaCartao.getNrContaCartao());
                        plastico.setDtVencimentoPlastico(GeraDados.geraDataVencCartao());

        plastico.setDtGeracaoCartao(GeraDados.geraDataInicioRelacionamento());
                        plastico.setNomeCliente(cliente.getNomeCliente());
```

```
                        /* GRAVA DADOS NO BANCO*/
                        GravaBancoDados.gravaDadosCliente(cliente);
                        GravaBancoDados.gravaDadosContaCartao(contaCartao);
                        GravaBancoDados.gravaDadosPlastico(plastico);

                        /* MENSAGENS DE CONFIRMAÇÃO PARA O USUARIO*/
                        JOptionPane.showMessageDialog(campoNomePai, "MCI: " +
cliente.getMciCliente());
                        JOptionPane.showMessageDialog(campoNomePai, "Limite do Cartão
de Crédito R$: " + df.format(GeraLimites.limiteCartao(cliente)));

                        JOptionPane.showMessageDialog(campoNomePai, "Cadastre uma
senha para o cartão solicitado!");

                    } else if (selecaoContaCorrente.isSelected()){

                        ContaCorrente contaCorrente = new ContaCorrente();
                        Plastico plastico = new Plastico();

                        /* DADOS DA CONTA CORRENTE*/

        contaCorrente.setSaldoAtual(GeraLimites.limiteContaCorrente(cliente));
                        contaCorrente.setMciCliente(cliente.getMciCliente());
                        contaCorrente.setNrConta(cliente.getNrContaCliente());

        contaCorrente.setDtInicioConta(GeraDados.geraDataInicioRelacionamento());

                        /* DADOS DO PLASTICO*/
                        plastico.setMciCliente(cliente.getMciCliente());
                        plastico.setNrPlastico(GeraDados.geraNrPlastico());
                        plastico.setNrContaCorrente(cliente.getNrContaCliente());
                        plastico.setNrContaCartao(0);
                        plastico.setDtVencimentoPlastico(GeraDados.geraDataVencCartao());

        plastico.setDtGeracaoCartao(GeraDados.geraDataInicioRelacionamento());
                        plastico.setNomeCliente(cliente.getNomeCliente());

                        /* GRAVA DADOS NO BANCO*/
                        GravaBancoDados.gravaDadosCliente(cliente);
                        GravaBancoDados.gravaDadosContaCorrente(contaCorrente);
                        GravaBancoDados.gravaDadosPlastico(plastico);

                        /* MENSAGENS DE CONFIRMAÇÃO PARA O USUARIO*/
                        JOptionPane.showMessageDialog(campoNomePai, "MCI: " +
cliente.getMciCliente());
                        JOptionPane.showMessageDialog(campoNomePai, "Nr. Conta: " +
cliente.getNrContaCliente());
                        JOptionPane.showMessageDialog(campoNomePai, "Limite da Conta
Corrente R$:" + df.format(GeraLimites.limiteContaCorrente(cliente)));
```

```
                        JOptionPane.showMessageDialog(campoNomePai, "Cadastre uma
senha para a conta solicitada!");

                    }

                } catch (SQLException ex) {

                    Logger.getLogger(Cadastro.class.getName()).log(Level.SEVERE, null,
ex);
                    labelAlerta1.setVisible(true);
                      labelAlerta1.setForeground(Color.red);
                      labelAlerta1.setText("PROBLEMA AO GRAVAR OS DADOS,
REPITA A OPERAÇÃO!");
                    return;

                }

            }

            }else{

                    labelAlerta1.setVisible(true);
            labelAlerta1.setForeground(Color.red);
            labelAlerta1.setText("ESCOLHA A OPERAÇÃO DESEJADA!");
            return;

            }

    }

    private void botaoGravarSenhaActionPerformed(java.awt.event.ActionEvent evt) {


        if(botaoGravarCadatro.isEnabled()){

                labelAlerta1.setVisible(true);
        labelAlerta1.setForeground(Color.red);
        labelAlerta1.setText("PREENCHA PRIMEIRAMENTE O CADASTRO!");
        return;

        }else{

                String senhaString  = campoSenha.getText();
            String confirmarSenhaString = campoConfirmarSenha.getText();

            if(campoSenha.getText() == null || ("").equals(campoSenha.getText())){

                labelAlerta1.setVisible(true);
```

```
                labelAlerta1.setForeground(Color.red);
                labelAlerta1.setText("DIGITE UMA SENHA COM 6 DÍGITOS!");
                return;
            }

            if(senhaString.length() != 6){

                labelAlerta1.setVisible(true);
                labelAlerta1.setForeground(Color.red);
                labelAlerta1.setText("A SENHA DEVE TER 6 DÍGITOS!");
                campoSenha.setText("");
                return;

            }

             if(campoConfirmarSenha.getText() == null ||
("").equals(campoConfirmarSenha.getText())){

                labelAlerta1.setVisible(true);
                labelAlerta1.setForeground(Color.red);
                labelAlerta1.setText("CONFIRME A SENHA!");
                return;
            }

            if(confirmarSenhaString.length() != 6){

                labelAlerta1.setVisible(true);
                labelAlerta1.setForeground(Color.red);
                labelAlerta1.setText("A SENHA DEVE TER 6 DÍGITOS!");
                campoConfirmarSenha.setText("");
                return;

            }

            char[] senha  = campoSenha.getPassword();
            char[] confirmaSenha = campoConfirmarSenha.getPassword();

            StringBuilder senhaDescompilada = new StringBuilder();
            StringBuilder confirmaSenhaDescompilada = new StringBuilder();

            for(int i = 0; i <= 5; i++){

              senhaDescompilada.append(senha[i]);
              confirmaSenhaDescompilada.append(confirmaSenha[i]);

            }

            String senhaFinal = senhaDescompilada.toString();
            String confirmaSenhaFinal = confirmaSenhaDescompilada.toString();
```

```
            if((senhaFinal).equals(confirmaSenhaFinal)){

                int opcao = JOptionPane.showConfirmDialog(campoNomePai, "Confirma a
gravação dos dados?",
                    "Confirmação de Senha", JOptionPane.YES_NO_CANCEL_OPTION);
                boolean flag;

                flag = opcao == JOptionPane.YES_OPTION;

                if( flag ){

                  campoSenha.setEditable(false);
                  campoConfirmarSenha.setEditable(false);
                  labelAlerta1.setVisible(false);
                  cliente.setSenhaCliente(senhaFinal);

                  try {

                     GravaBancoDados.gravaSenhaCliente(cliente);

                  } catch (SQLException e) {

                     e.printStackTrace();

                  }


                  JOptionPane.showMessageDialog(campoNomePai, "Senhas cadastradas
com sucesso!");
                  botaoGravarSenha.setEnabled(false);
                }

            } else{

               campoSenha.setEditable(true);
               campoConfirmarSenha.setEditable(true);
               labelAlerta1.setVisible(true);
               labelAlerta1.setForeground(Color.red);
               labelAlerta1.setText("SENHAS NÃO CONFEREM!");
               campoSenha.setText("");
               campoConfirmarSenha.setText("");
               return;

            }

        }

    }
```

```
    private void subMenuArquivoSairActionPerformed(java.awt.event.ActionEvent evt) {

        System.exit(0);

    }

    private void botaoMenuSairMouseClicked(java.awt.event.MouseEvent evt) {

        System.exit(0);

    }

    private void botaoFinalizarActionPerformed(java.awt.event.ActionEvent evt) {

            //rotina grava PIN no cartao
            CompilaArquivoBat cab = new CompilaArquivoBat();
            cab.carregaArquivoBat("CompilarExecuta_GravaPIN");

            JOptionPane.showMessageDialog(campoNomePai, "Cadastro Finalizado com
sucesso!");
            System.exit(0);

    }

    public void selecaoContaCorrenteActionPerformed(java.awt.event.ActionEvent evt) {

            //inicializa leitora ID
            bsp = new NBioBSPJNI();
            objIndexSearch = bsp.new IndexSearch();

            //rotina que compila a classe applet e converte para arquivo CAP
            CompilaArquivoBat cab = new CompilaArquivoBat();
            cab.carregaArquivoBat("Compila_ClassGravaRecuperaPIN");

            //desabilita escolher a outra operação
        selecaoCartaoCredito.setEnabled(false);
        if(selecaoContaCorrente.isSelected()){

            selecaoCartaoCredito.setEnabled(false);
        }else{

            selecaoCartaoCredito.setEnabled(true);
        }
    }

    public void selecaoCartaoCreditoActionPerformed(java.awt.event.ActionEvent evt) {

            //inicializa leitora ID
```

```
        bsp = new NBioBSPJNI();
        objIndexSearch = bsp.new IndexSearch();

        //compila a classe applet e converte para arquivo CAP
        CompilaArquivoBat cab = new CompilaArquivoBat();
        cab.carregaArquivoBat("Compila_ClassGravaRecuperaPIN");

        //desabilita escolher a outra operação
    selecaoContaCorrente.setEnabled(false);
    if(selecaoCartaoCredito.isSelected()){

        selecaoContaCorrente.setEnabled(false);

    }else{

        selecaoContaCorrente.setEnabled(true);

    }
}


public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Cadastro().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JMenuBar barraMenu;
private javax.swing.JButton botaoFinalizar;
public javax.swing.JButton botaoGravarCadatro;
private javax.swing.JButton botaoGravarSenha;
private javax.swing.JMenu botaoMenuArquivo;
private javax.swing.JMenu botaoMenuSair;
private javax.swing.JButton botaoRecolherDigital;
private javax.swing.JComboBox caixaNaturezaOcupacao;
private javax.swing.JFormattedTextField campoCPF;
private javax.swing.JPasswordField campoConfirmarSenha;
private javax.swing.JTextField campoEndereco;
private javax.swing.JTextField campoIdentidade;
private javax.swing.JTextField campoNomeMae;
private javax.swing.JTextField campoNomePai;
private javax.swing.JTextField campoNomeTitular;
private javax.swing.JTextField campoOcupacao;
public javax.swing.JFormattedTextField campoRenda;
private javax.swing.JPasswordField campoSenha;
private javax.swing.JFormattedTextField campoTelefone;
```

```
    private javax.swing.JLabel labelAlerta;
    private javax.swing.JLabel labelAlerta1;
    private javax.swing.JLabel labelCPF;
    private javax.swing.JLabel labelConfirmarSenha;
    private javax.swing.JLabel labelEndereco;
    private javax.swing.JLabel labelIdentidade;
    private javax.swing.JLabel labelNaturezaOcupacao;
    private javax.swing.JLabel labelNomeMae;
    private javax.swing.JLabel labelNomePai;
    private javax.swing.JLabel labelNomeTitular;
    private javax.swing.JLabel labelOcupacao;
    private javax.swing.JLabel labelOpcaoOperacao;
    private javax.swing.JLabel labelRenda;
    private javax.swing.JLabel labelSenha;
    private javax.swing.JLabel labelTelefone;
    private javax.swing.JRadioButton selecaoCartaoCredito;
    private javax.swing.JRadioButton selecaoContaCorrente;
    private javax.swing.JSeparator separadorCadastro;
    private javax.swing.JSeparator separadorDigital;
    private javax.swing.JSeparator separadorOperacao;
    private javax.swing.JSeparator separadorSenha;
    private javax.swing.JRadioButtonMenuItem subMenuArquivoSair;
    // End of variables declaration

}
```

Transcreve-se abaixo código da interface do aplicativo de simulação do terminal POS:

```
package projetofinal;

import javax.swing.JOptionPane;

import br.com.vinicius.Cliente;
import br.com.vinicius.action.Contabilizacao;
import br.com.vinicius.action.GeraDados;
import br.com.vinicius.smartcard.CompilaArquivoBat;

import com.nitgen.SDK.BSP.NBioBSPJNI;


/**
 *
 * @author Vinícius
 */
public class Compra extends javax.swing.JFrame {
```

```
private static final long serialVersionUID = -6036193053800206910L;

private NBioBSPJNI bsp;
private NBioBSPJNI.DEVICE_ENUM_INFO deviceEnumInfo;
private int mci;
String temp = "0";
String textoOpcaoOperacao = "1 - Débito 2 - Crédito";
String textoCompraNaoIniciada = "Inicie uma nova compra...";
String opcaoEscolhida;
String resultadoCompra = null;
int sim = 1;

/** Cria novo form Compra */
public Compra() {
        initComponents();
}

private void initComponents() {

        botaoNovaCompra = new javax.swing.JButton();
        botaoCancelar = new javax.swing.JButton();
        botaoCorrigir = new javax.swing.JButton();
        botaoFinalizar = new javax.swing.JButton();
        botao1 = new javax.swing.JButton();
        botao2 = new javax.swing.JButton();
        botao3 = new javax.swing.JButton();
        botao4 = new javax.swing.JButton();
        botao5 = new javax.swing.JButton();
        botao6 = new javax.swing.JButton();
        botao7 = new javax.swing.JButton();
        botao8 = new javax.swing.JButton();
        botao9 = new javax.swing.JButton();
        botao0 = new javax.swing.JButton();
        botaoVelha = new javax.swing.JButton();
        botaoEstrela = new javax.swing.JButton();
        separador = new javax.swing.JSeparator();
        campoValor = new javax.swing.JFormattedTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setResizable(false);

        botaoNovaCompra.setText("NOVA COMPRA");
        botaoNovaCompra.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                        botaoNovaCompraActionPerformed(evt);
                }
        });

        botaoCancelar.setBackground(new java.awt.Color(255, 0, 0));
```

```
botaoCancelar.setText("Cancelar");
botaoCancelar.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoCancelarActionPerformed(evt);
        }
});

botaoCorrigir.setBackground(new java.awt.Color(255, 255, 0));
botaoCorrigir.setText("Corrigir");
botaoCorrigir.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoCorrigirActionPerformed(evt);
        }
});

botaoFinalizar.setBackground(new java.awt.Color(0, 153, 51));
botaoFinalizar.setText("Finalizar");
botaoFinalizar.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoFinalizarActionPerformed(evt);
        }
});

botao1.setText("1");
botao1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao1ActionPerformed(evt);
        }
});

botao2.setText("2");
botao2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao2ActionPerformed(evt);
        }
});

botao3.setText("3");
botao3.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao3ActionPerformed(evt);
        }
});

botao4.setText("4");
botao4.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao4ActionPerformed(evt);
        }
```

```
});

botao5.setText("5");
botao5.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao5ActionPerformed(evt);
        }
});

botao6.setText("6");
botao6.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao6ActionPerformed(evt);
        }
});

botao7.setText("7");
botao7.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao7ActionPerformed(evt);
        }
});

botao8.setText("8");
botao8.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao8ActionPerformed(evt);
        }
});

botao9.setText("9");
botao9.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao9ActionPerformed(evt);
        }
});

botao0.setText("0");
botao0.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botao0ActionPerformed(evt);
        }
});

botaoVelha.setText("# .");
botaoVelha.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoVelhaActionPerformed(evt);
        }
```

```
        });

        botaoEstrela.setText("*");
        botaoEstrela.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                botaoEstrelaActionPerformed(evt);
            }
        });

        campoValor.setBackground(new java.awt.Color(102, 153, 0));
        campoValor.setBorder(javax.swing.BorderFactory.createLineBorder(
                new java.awt.Color(0, 0, 0), 3));
        campoValor
                .setFormatterFactory(new
javax.swing.text.DefaultFormatterFactory(
                        new javax.swing.text.NumberFormatter(
                                new
java.text.DecimalFormat("¤#,##0.00"))));
        campoValor.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        campoValor.setFont(new java.awt.Font("Courier New", 1, 18)); // NOI18N

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
                getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(layout
                .createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
                .addGroup(
                        layout.createSequentialGroup()
                                .addContainerGap()
                                .addGroup(

    layout.createParallelGroup(

    javax.swing.GroupLayout.Alignment.LEADING)
                                                                    .add
Group(

        layout.createParallelGroup(

                javax.swing.GroupLayout.Alignment.TRAILING)

                .addGroup(

                        layout.createSequentialGroup()

                                .addComponent(

                                        botao4)
```

```
                .addGap(18,
                        18,
                        18)
                .addComponent(
                        botao5)
                .addGap(18,
                        18,
                        18)
                .addComponent(
                        botao6))
        .addGroup(
                layout.createSequentialGroup()
                .addComponent(
                        botao1)
                .addGap(18,
                        18,
                        18)
                .addComponent(
                        botao2)
                .addGap(18,
                        18,
                        18)
                .addComponent(
                        botao3)))
                                .add
```

```
Group(
                layout.createSequentialGroup()
                        .addComponent(
                                botao7)
                        .addGap(18, 18,
                                18)
                        .addComponent(
                                botao8)
                        .addGap(18, 18,
                                18)
                        .addComponent(
                                botao9))
                                                                .add
Group(
                javax.swing.GroupLayout.Alignment.TRAILING,
                layout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.TRAILING)
                        .addGroup(
                                javax.swing.GroupLayout.Alignment.LEADING,
                                layout.createSequentialGroup()
                                        .addComponent(
                                                botaoEstrela)
                                        .addGap(18,
                                                18,
                                                18)
                                        .addComponent(
```

```
                botao0)
            .addGap(18,
                18,
                18)
            .addComponent(
                botaoVelha,
javax.swing.GroupLayout.PREFERRED_SIZE,
                42,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGroup(
            javax.swing.GroupLayout.Alignment.LEADING,
            layout.createSequentialGroup()
                .addComponent(
                    botaoCancelar)
                .addGap(18,
                    18,
                    18)
                .addComponent(
                    botaoCorrigir)
                .addGap(18,
                    18,
                    18)
                .addComponent(
```

```
                                                                botaoFinalizar))))
                                                        .addContainerGap())
                                .addComponent(separador,
javax.swing.GroupLayout.DEFAULT_SIZE,
                                                281, Short.MAX_VALUE)
                                .addGroup(
                                                layout.createSequentialGroup()
                                                        .addContainerGap(93,
Short.MAX_VALUE)
                                                        .addComponent(botaoNovaCompr
a)
                                                        .addContainerGap(93,
Short.MAX_VALUE))
                                .addGroup(
                                                layout.createSequentialGroup()
                                                        .addContainerGap()
                                                        .addComponent(campoValor,

        javax.swing.GroupLayout.DEFAULT_SIZE,
                                                                261,
Short.MAX_VALUE).addContainerGap()));

                layout.linkSize(javax.swing.SwingConstants.HORIZONTAL,
                                new java.awt.Component[] { botao0, botao1, botao2, botao3,
                                                botao4, botao5, botao6, botao7, botao8, botao9,
                                                botaoCancelar, botaoCorrigir, botaoEstrela,
                                                botaoFinalizar, botaoVelha });

                layout.setVerticalGroup(layout
                                .createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
                                .addGroup(
                                                layout.createSequentialGroup()
                                                        .addGap(16, 16, 16)
                                                        .addComponent(botaoNovaCompr
a)
                                                        .addGap(18, 18, 18)
                                                        .addComponent(separador,

        javax.swing.GroupLayout.PREFERRED_SIZE,
                                                                10,

        javax.swing.GroupLayout.PREFERRED_SIZE)
                                                        .addPr
eferredGap(

        javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                        .addComponent(campoValor,
```

```
        javax.swing.GroupLayout.PREFERRED_SIZE,
                                                                    161,

        javax.swing.GroupLayout.PREFERRED_SIZE)
                                                        .addGap(18, 18, 18)
                                                        .addGroup(

        layout.createParallelGroup(

        javax.swing.GroupLayout.Alignment.BASELINE)
                                                                    .add
Component(botao1)
                                                                    .add
Component(botao2)
                                                                    .add
Component(botao3))
                                                        .addPreferredGap(

        javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                        .addGroup(

        layout.createParallelGroup(

        javax.swing.GroupLayout.Alignment.BASELINE)
                                                                    .add
Component(botao4)
                                                                    .add
Component(botao5)
                                                                    .add
Component(botao6))
                                                        .addGap(11, 11, 11)
                                                        .addGroup(

        layout.createParallelGroup(

        javax.swing.GroupLayout.Alignment.BASELINE)
                                                                    .add
Component(botao7)
                                                                    .add
Component(botao8)
                                                                    .add
Component(botao9))
                                                        .addPreferredGap(

        javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                                        .addGroup(

        layout.createParallelGroup(
```

```
        javax.swing.GroupLayout.Alignment.BASELINE)
                                                                    .add
Component(botaoEstrela)
                                                                    .add
Component(
                botao0,
                javax.swing.GroupLayout.PREFERRED_SIZE,
                23,
                javax.swing.GroupLayout.PREFERRED_SIZE)
                                                                    .add
Component(botaoVelha))
                                                        .addGap(18, 18, 18)
                                                        .addGroup(
        layout.createParallelGroup(
        javax.swing.GroupLayout.Alignment.BASELINE)
                                                                    .add
Component(botaoCancelar)
                                                                    .add
Component(botaoFinalizar)
                                                                    .add
Component(botaoCorrigir))
                                                        .addContainerGap(51,
Short.MAX_VALUE)));
                layout.linkSize(
                        javax.swing.SwingConstants.VERTICAL,
                        new java.awt.Component[] { botao1, botao2, botao3, botao4,
                                botao5, botao6, botao7, botao8, botao9,
botaoCancelar,
                                botaoCorrigir, botaoEstrela, botaoFinalizar,
botaoVelha });
                pack();
        }
        private void botaoNovaCompraActionPerformed(java.awt.event.ActionEvent evt) {
                //abrir leitora do cartão,
                bsp = new NBioBSPJNI();
                bsp.OpenDevice();
```

```
            JOptionPane.showMessageDialog(rootPane, "Insira o cartão na leitora!");

            //rotina que recupera informacoes do cartao
        CompilaArquivoBat cab = new CompilaArquivoBat();
        cab.carregaArquivoBat("CompilarExecutar_RecuperaPIN");

            mci = recuperaMciTransacao();

            campoValor.setText(textoOpcaoOperacao);
            botaoNovaCompra.setEnabled(false);
            opcaoEscolhida = "escolher";

        }

        private void botaoFinalizarActionPerformed(java.awt.event.ActionEvent evt) {

            int indicador = 0;

            indicador = verificaDigital(mci);

            Cliente cliente = new Cliente();
            cliente.setMciCliente(mci);

            if(indicador == 1){

                if(opcaoEscolhida == "debito"){

                    Double valorCompraDebito =
Double.parseDouble(campoValor.getText());

                    campoValor.setText("Processando...");

                    //delay
                    try {
                        Thread.sleep(5000);

                    }catch (Exception e) {
                        e.printStackTrace();
                    }

                    try {
                        Contabilizacao contabil = new Contabilizacao();
                        resultadoCompra = contabil.contabilDebito(cliente,
valorCompraDebito);
                        campoValor.setText(resultadoCompra);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
```

```
                }

                if(opcaoEscolhida == "credito"){

                        Double valorCompraCredito =
Double.parseDouble(campoValor.getText());

                        campoValor.setText("Processando...");

                        //delay
                        try {
                                Thread.sleep(5000);

                        }catch (Exception e) {
                                e.printStackTrace();
                        }

                        try {
                                Contabilizacao contabil = new Contabilizacao();
                                resultadoCompra = contabil.contabilCredito(cliente,
valorCompraCredito);
                                campoValor.setText(resultadoCompra);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }

                }

        // se a variavel for de insucesso
        }else{

                campoValor.setText("Processando...");

                //delay
                try {
                        Thread.sleep(5000);
                }catch (Exception e) {
                        e.printStackTrace();
                }

                try {
                        Contabilizacao contabil = new Contabilizacao();
                        resultadoCompra = contabil.bloqueiaCartão(cliente);
                        campoValor.setText(resultadoCompra);
                } catch (Exception e) {
                        e.printStackTrace();
                }
```

```
        }

}

private void botaoCorrigirActionPerformed(java.awt.event.ActionEvent evt) {

        if(campoValor.getText().equals(textoCompraNaoIniciada)){
                return;
        }
        else if(campoValor.getText().equals(textoOpcaoOperacao)){
                return;
        }
        else{
                temp = temp.replace(temp.substring(temp.length() - 1), "");
                campoValor.setText(temp);
        }

}

private void botaoCancelarActionPerformed(java.awt.event.ActionEvent evt) {
        campoValor.setText("");
        JOptionPane.showMessageDialog(rootPane, "Operação Cancelada!");
        botaoNovaCompra.setEnabled(true);
}

private void botaoVelhaActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado(".");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao0ActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("0");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
```

```
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botaoEstrelaActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("*");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao9ActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("9");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao8ActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("8");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao7ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("7");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao6ActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("6");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao5ActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("5");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao4ActionPerformed(java.awt.event.ActionEvent evt) {
        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("4");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
```

```
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao3ActionPerformed(java.awt.event.ActionEvent evt) {

        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "debito" || opcaoEscolhida == "credito") {
                        numeroDigitado("3");
                } else {
                        campoValor.setText(textoOpcaoOperacao);
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);

        }
}

private void botao2ActionPerformed(java.awt.event.ActionEvent evt) {

        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "escolher") {
                        opcaoEscolhida = "credito";
                        campoValor.setText("");
                        JOptionPane.showMessageDialog(rootPane, "Digite valor:");

                } else {
                        numeroDigitado("2");
                }
        } else {
                campoValor.setText(textoCompraNaoIniciada);
        }

}

private void botao1ActionPerformed(java.awt.event.ActionEvent evt) {

        if (botaoNovaCompra.isEnabled() == false) {

                if (opcaoEscolhida == "escolher") {
                        opcaoEscolhida = "debito";
                        campoValor.setText("");
                        JOptionPane.showMessageDialog(rootPane, "Digite valor:");

                } else {
```

```
                        numeroDigitado("1");
                    }
                } else {
                    campoValor.setText(textoCompraNaoIniciada);
                }
        }

        public static void main(String args[]) {
                java.awt.EventQueue.invokeLater(new Runnable() {
                    public void run() {
                        new Compra().setVisible(true);
                    }
                });
        }

        private javax.swing.JButton botao0;
        private javax.swing.JButton botao1;
        private javax.swing.JButton botao2;
        private javax.swing.JButton botao3;
        private javax.swing.JButton botao4;
        private javax.swing.JButton botao5;
        private javax.swing.JButton botao6;
        private javax.swing.JButton botao7;
        private javax.swing.JButton botao8;
        private javax.swing.JButton botao9;
        private javax.swing.JButton botaoCancelar;
        private javax.swing.JButton botaoCorrigir;
        private javax.swing.JButton botaoEstrela;
        private javax.swing.JButton botaoFinalizar;
        private
javax.swing.JButton botaoNovaCompra;
        private javax.swing.JButton botaoVelha;
        private javax.swing.JFormattedTextField campoValor;
        private javax.swing.JSeparator separador;

        public void numeroDigitado(String numero) {
                temp = campoValor.getText();
                temp = temp + (numero);
                campoValor.setText(temp);

        }

        public int verificaDigital(int mci){

                //variável de controle
                int ind = 0;

                //variável utilizada pelo software da Nitgen durante a comparação
                NBioBSPJNI.INPUT_FIR inputFIR = bsp.new INPUT_FIR();
```

```
                //variável de confirmação
                Boolean bResult = new Boolean(false);

                //variável que armazena novo exemplar
                NBioBSPJNI.FIR_PAYLOAD payload = bsp.new FIR_PAYLOAD();

                // variável para armazenamento do template gravado no banco de dados
                NBioBSPJNI.FIR_TEXTENCODE textSavedFIR;
                textSavedFIR = bsp.new FIR_TEXTENCODE();

                //recupera digital armazenada no banco de dados de acordo com MCI passado
como parämetro
                try {
                        textSavedFIR.TextFIR = GeraDados.recuperaID(mci);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                //atribui valor recebido do banco de dados
                inputFIR.SetTextFIR(textSavedFIR);

                //aciona software para comparação
                bsp.Verify(inputFIR, bResult, payload);

                //fecha dispositivo
                bsp.CloseDevice();

                //verifica se houve erro
                if (bsp.IsErrorOccured() == false){

                        if (bResult){
                                //ID compatível
                                ind = 1;
                        }
                        else{
                                //ID incompatível
                                ind = 0;
                        }
                }
                return ind;
        }


        public int recuperaMciTransacao(){

                int mciTransacao = 0;
        Connection con = null;
        PreparedStatement ps = null;
```

```
		ResultSet rs = null;

	try{
			con = Conexao.estabeleceConexao();
			String sqlQuery = null;
			sqlQuery = "SELECT MCI FROM mci_transacao";
			ps = con.prepareStatement(sqlQuery);
			rs = ps.executeQuery();

			if(rs.next()){
				mciTransacao = rs.getInt("MCI");
			}
			ps.close();
			rs.close();
		}
		catch(SQLException e){
			e.printStackTrace();
		}
		Conexao.fecharConexao();
		return mciTransacao;
	}


}
```

Transcreve-se abaixo código da classe Java responsável por gravar os dados da aplicação em banco de dados:

```
package br.com.vinicius.action;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import br.com.vinicius.Cliente;
import br.com.vinicius.Conexao;
import br.com.vinicius.ContaCorrente;
import br.com.vinicius.ContaCartao;
import br.com.vinicius.Plastico;

public class GravaBancoDados {

   public static void gravaDadosCliente(Cliente cliente) throws SQLException{

      Connection con = null;
```

```
        PreparedStatement ps = null;

        try{

            con = Conexao.estabeleceConexao();

            StringBuilder sql = new StringBuilder();

            sql.append(" INSERT INTO clientes (MCI, CPF, IDENTIDADE, NR_CONTA,");
            sql.append(" NM_TITULAR, NM_MAE, NM_PAI, OCUPACAO, NT_OCUPACAO,
RENDA_TITULAR, TEL,");
            sql.append(" RESIDENCIA, SENHA, FINGER_PRINT, DT_INICIO, DT_FIM)");
            sql.append(" VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");

            ps = con.prepareStatement(sql.toString());

            ps.setInt(1, cliente.getMciCliente());
            ps.setString(2, cliente.getCpfCliente());
            ps.setString(3, cliente.getIdCliente());
            ps.setInt(4, cliente.getNrContaCliente());
            ps.setString(5, cliente.getNomeCliente());
            ps.setString(6, cliente.getNomeMaeCliente());
            ps.setString(7, cliente.getNomePaiCliente());
            ps.setString(8, cliente.getOcupacaoCliente());
            ps.setString(9, cliente.getNatOcupacaoCliente());
            ps.setString(10, cliente.getRendaCliente());
            ps.setString(11, cliente.getTelefoneCliente());
            ps.setString(12, cliente.getResidenciaCliente());
            ps.setString(13, "0");
            ps.setString(14, "");
            ps.setString(15, cliente.getDtInicioRelCliente());
            ps.setString(16, "2020-12-31");

            ps.executeUpdate();

            ps.close();

        }

        catch(Exception e){

            e.printStackTrace();

        }

        Conexao.fecharConexao();
    }
```

```
public static void gravaSenhaCliente(Cliente cliente) throws SQLException{

	Connection con = null;
	PreparedStatement ps = null;

	try{

		con = Conexao.estabeleceConexao();

		String sql = null;
		sql = " UPDATE clientes SET SENHA = ? WHERE MCI = ?";

		ps = con.prepareStatement(sql);
		ps.setString(1, cliente.getSenhaCliente());
		ps.setInt(2, cliente.getMciCliente());

		ps.executeUpdate();

		ps.close();

	}catch(Exception e){

		 e.printStackTrace();

	}

}

public static void gravaImpressaoDigital(Cliente cliente) throws SQLException{

	Connection con = null;
	PreparedStatement ps = null;

	try{

		con = Conexao.estabeleceConexao();

		String sql = null;
		sql = " UPDATE clientes SET FINGER_PRINT = ? WHERE MCI = ?";

		ps = con.prepareStatement(sql);
		ps.setString(1, cliente.getFingerPrintCliente());
		ps.setInt(2, cliente.getMciCliente());

		ps.executeUpdate();

		ps.close();

	}catch(Exception e){
```

```
                e.printStackTrace();

        }

    }

    public static void gravaDadosPlastico(Plastico plastico) throws SQLException{

        Connection con = null;
        PreparedStatement ps = null;

        try{

            con = Conexao.estabeleceConexao();

            StringBuilder sql = new StringBuilder();

            sql.append(" INSERT INTO plastico (NR_PLASTICO,
conta_cartao_NR_CONTA_CARTAO,");
            sql.append(" clientes_MCI, conta_NUM_CONTA, DT_VENCIMENTO,
NM_TITULAR,");
            sql.append(" BANDEIRA, BLOQUEADO, DT_GERACAO, QNT_ERROS, ATR)");
            sql.append(" values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");

            ps = con.prepareStatement(sql.toString());

            ps.setInt(1, plastico.getNrPlastico());
            ps.setInt(2, plastico.getNrContaCartao());
            ps.setInt(3, plastico.getMciCliente());
            ps.setInt(4, plastico.getNrContaCorrente());
            ps.setString(5, plastico.getDtVencimentoPlastico());
            ps.setString(6, plastico.getNomeCliente());
            ps.setString(7, "CardVini");
            ps.setString(8, "N");
            ps.setString(9, plastico.getDtGeracaoCartao());
            ps.setInt(10, 0);
            ps.setString(11, "3BF81800008131FE450073C8400000900080");

            ps.executeUpdate();

            ps.close();

        }

        catch(SQLException e){

            e.printStackTrace();
```

```
        }

        Conexao.fecharConexao();

    }

    public static void gravaDadosContaCorrente(ContaCorrente contaCorrente) throws
SQLException{

        Connection con = null;
        PreparedStatement ps = null;

        try{

            con = Conexao.estabeleceConexao();

            StringBuilder sql = new StringBuilder();

            sql.append(" INSERT INTO conta_corrente (NUM_CONTA, clientes_MCI,
AGENCIA,");
            sql.append(" SD_ANTERIOR, SD_ATUAL, VL_ULT_TRNS, DT_ULT_TRNS,");
            sql.append(" LCL_ULT_TRNS, TP_ATNTC, DT_INICIO, DT_FIM)");
            sql.append(" values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");

            ps = con.prepareStatement(sql.toString());

            ps.setInt(1, contaCorrente.getNrConta());
            ps.setInt(2, contaCorrente.getMciCliente());
            ps.setString(3, "1234-5");
            ps.setString(4, "0");
            ps.setFloat(5, contaCorrente.getSaldoAtual());
            ps.setString(6, "0");
            ps.setString(7, "0000-00-00");
            ps.setString(8, "0");
            ps.setString(9, "0");
            ps.setString(10, contaCorrente.getDtInicioConta());
            ps.setString(11, "2020-12-31");

            ps.executeUpdate();

            ps.close();

        }

        catch (Exception e){

            e.printStackTrace();

        }
```

```
        Conexao.fecharConexao();

    }

    public static void gravaDadosContaCartao(ContaCartao contaCartao) throws
SQLException{

            Connection con = null;
        PreparedStatement ps = null;

        try{

            con = Conexao.estabeleceConexao();

            StringBuilder sql = new StringBuilder();

            sql.append(" INSERT INTO conta_cartao (NR_CONTA_CARTAO, clientes_MCI,
BANDEIRA,");
            sql.append(" LMT_ANTERIOR, LMT_ATUAL, VLR_ULT_TRNS,
DT_ULT_TRNS,");
            sql.append(" LCL_ULT_TRNS, TP_ATNTC, DT_INICIO, DT_FIM)");
            sql.append(" values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");

            ps = con.prepareStatement(sql.toString());

            ps.setInt(1, contaCartao.getNrContaCartao());
            ps.setInt(2, contaCartao.getMciCliente());
            ps.setString(3, "CardVini");
            ps.setString(4, "0");
            ps.setFloat(5, contaCartao.getLimiteAtual());
            ps.setString(6, "0");
            ps.setString(7, "0000-00-00");
            ps.setString(8, "0");
            ps.setString(9, "0");
            ps.setString(10, contaCartao.getDtInicioContaCartao());
            ps.setString(11, "2020-12-31");

            ps.executeUpdate();

            ps.close();

        }

        catch (Exception e){

            e.printStackTrace();

        }
```

```
        Conexao.fecharConexao();
    }

}
```

Transcreve-se abaixo código da classe Java responsável pela geração dos dados da aplicação:

```
package br.com.vinicius.action;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Calendar;
import java.sql.PreparedStatement;

import br.com.vinicius.Conexao;
import java.text.SimpleDateFormat;
import java.util.Date;

public class GeraDados{

    public static int geraNrContaCorrente() throws SQLException {

        int nrContaCorrente = 0;

        Connection con = null;
        PreparedStatement ps = null;
        ResultSet rs = null;

        try{

            con = Conexao.estabeleceConexao();

            StringBuilder sql = new StringBuilder();

            sql.append("SELECT MAX(NR_CONTA)");
            sql.append("FROM clientes");

            ps = con.prepareStatement(sql.toString());
            rs = ps.executeQuery();

            if(rs.next()){

                nrContaCorrente = rs.getInt("MAX(NR_CONTA)") + 1;
```

```
        }

        ps.close();
        rs.close();

    }

    catch(SQLException e){

        e.printStackTrace();

    }

    Conexao.fecharConexao();

    return nrContaCorrente;

}

public static int geraMCICliente() throws SQLException{

    int mciCliente = 0;

    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try{

        con = Conexao.estabeleceConexao();

        StringBuilder sql = new StringBuilder();

        sql.append("SELECT MAX(MCI)");
        sql.append("FROM clientes");

        ps = con.prepareStatement(sql.toString());
        rs = ps.executeQuery();

        if(rs.next()){

            mciCliente = rs.getInt("MAX(MCI)") + 1;

        }

        ps.close();
        rs.close();

    }
```

```
    catch(SQLException e){

        e.printStackTrace();

    }

    Conexao.fecharConexao();

    return mciCliente;
}

public static int geraNrContaCartao() throws SQLException{

    int nrContaCartao = 0;

    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try{

        con = Conexao.estabeleceConexao();

        StringBuilder sql = new StringBuilder();

        sql.append("SELECT MAX(NR_CONTA_CARTAO)");
        sql.append("FROM conta_cartao");

        ps = con.prepareStatement(sql.toString());
        rs = ps.executeQuery();

        if(rs.next()){

            nrContaCartao = rs.getInt("MAX(NR_CONTA_CARTAO)") + 1;

        }

        ps.close();
        rs.close();

    }

    catch(SQLException e){

        e.printStackTrace();

    }
```

```
    Conexao.fecharConexao();

    return nrContaCartao;

}

 public static int geraNrPlastico() throws SQLException{

    int nrPlastico = 0;

    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try{

       con = Conexao.estabeleceConexao();

       StringBuilder sql = new StringBuilder();

       sql.append("SELECT MAX(NR_PLASTICO)");
       sql.append("FROM plastico");

       ps = con.prepareStatement(sql.toString());
       rs = ps.executeQuery();

       if(rs.next()){

            nrPlastico = rs.getInt("MAX(NR_PLASTICO)") + 1;

       }

       ps.close();
       rs.close();

    }

    catch(SQLException e){

       e.printStackTrace();

    }

    Conexao.fecharConexao();

    return nrPlastico;

 }
```

```
public static String recuperaID(int mci) throws SQLException{

    String fingerPrint = null;

    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try{

        con = Conexao.estabeleceConexao();

        StringBuilder sql = new StringBuilder();

        sql.append(" SELECT FINGER_PRINT FROM clientes");
        sql.append(" WHERE MCI = ?");

        ps = con.prepareStatement(sql.toString());
        ps.setInt(1, mci);
        rs = ps.executeQuery();

        if(rs.next()){

                fingerPrint = rs.getString("FINGER_PRINT");

        }

        ps.close();
        rs.close();

    }

    catch(SQLException e){

        e.printStackTrace();

    }

    Conexao.fecharConexao();

    return fingerPrint;

}

public static String geraDataVencCartao() throws SQLException{

   SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
   Date hoje = new Date();
   Calendar calendar = Calendar.getInstance();
```

```
        calendar.setTime(hoje);
        calendar.add(Calendar.YEAR, 5);

        return sdf.format(calendar.getTime());

    }

    public static String geraDataInicioRelacionamento() {

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Date hoje = new Date();
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(hoje);

        return sdf.format(calendar.getTime());

    }

}
```

Transcreve-se abaixo código da classe Java responsável pelas contabilizações:

```
package br.com.vinicius.action;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import br.com.vinicius.Cliente;
import br.com.vinicius.Conexao;

public class Contabilizacao {

        private Double limiteAtual;
        private String bloqueado;
        private Date dataVencimento;
        private String codigoRetorno;
        private int nrContaCartao;
        private Double saldoAtual;
        private int nrContaCorrente;
```

```
public String contabilCredito(Cliente cliente, Double valorCompra) throws
SQLException {

        Connection connection = null;
        PreparedStatement ps = null;
        ResultSet rs = null;

        try {

                connection = Conexao.estabeleceConexao();

                StringBuilder query = new StringBuilder();

                query.append(" SELECT a.LMT_ATUAL, b.BLOQUEADO,
b.DT_VENCIMENTO, a.NR_CONTA_CARTAO");
                query.append(" FROM conta_cartao a INNER JOIN plastico b");
                query.append(" ON a.clientes_MCI = b.clientes_MCI");
                query.append(" WHERE a.clientes_MCI = ?");

                ps = connection.prepareStatement(query.toString());

                ps.setInt(1, cliente.getMciCliente());

                rs = ps.executeQuery();

                if(rs.next()){

                        limiteAtual = rs.getDouble("LMT_ATUAL");
                        bloqueado = rs.getString("BLOQUEADO");
                        dataVencimento = rs.getDate("DT_VENCIMENTO");
                        nrContaCartao = rs.getInt("NR_CONTA_CARTAO");

                        if( valorCompra > limiteAtual){

                                codigoRetorno = "Limite Insuficiente";
                                return codigoRetorno;

                        }

                        if( bloqueado.equals("S")){

                                codigoRetorno = "Cartão Bloqueado";
                                return codigoRetorno;
                        }

                        if (dataVencimento.compareTo(new Date()) < 0 ){

                                codigoRetorno = "Cartão Vencido";
                                return codigoRetorno;
```

```
                    }else {

                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-
MM-dd");
                    Date hoje = new Date();
                    Calendar calendar = Calendar.getInstance();
                    calendar.setTime(hoje);

                    String dataTransacao = sdf.format(calendar.getTime());

                        try {

                            StringBuilder sqlGrava = new StringBuilder();

                            sqlGrava.append(" UPDATE conta_cartao SET
LMT_ANTERIOR = ?,");
                            sqlGrava.append(" LMT_ATUAL = ?,
VLR_ULT_TRNS = ?, LCL_ULT_TRNS = ?,");
                            sqlGrava.append(" DT_ULT_TRNS = ?,
TP_ATNTC = ? WHERE clientes_MCI = ?");

                            ps =
connection.prepareStatement(sqlGrava.toString());

                            ps.setDouble(1, limiteAtual);
                            ps.setDouble(2, (limiteAtual - valorCompra));
                            ps.setDouble(3, valorCompra);
                            ps.setString(4, "Mercearia do Zé");
                            ps.setString(5, dataTransacao);
                            ps.setInt(6, 1);
                            ps.setInt(7, cliente.getMciCliente());

                            ps.executeUpdate();

                            try {

                                StringBuilder sqlGravaRegistro = new
StringBuilder();

                                sqlGravaRegistro.append(" INSERT
INTO reg_conta_cartao (VLR_TRNS, LCL_TRNS, DT_TRNS,");
                        sqlGravaRegistro.append(" TP_AUT,
conta_cartao_NR_CONTA_CARTAO)");
                        sqlGravaRegistro.append(" values (?, ?, ?, ?, ?)");

                                ps =
connection.prepareStatement(sqlGravaRegistro.toString());
```

```
                                        ps.setDouble(1, valorCompra);
                                        ps.setString(2, "Mercearia do Zé");
                                        ps.setString(3, dataTransacao);
                                        ps.setInt(4, 1);
                                        ps.setInt(5, nrContaCartao);

                                        ps.executeUpdate();

                                        codigoRetorno = "Aprovado";

                                } catch (SQLException e) {

                                        e.printStackTrace();
                                        codigoRetorno = "Erro - Tente
Novamente";
                                }


                        }

                        catch (SQLException e) {

                                e.printStackTrace();
                                codigoRetorno = "Erro - Tente Novamente";
                        }

                }


                }else{
                        codigoRetorno = "Operação Incorreta";
                }

        } catch (SQLException e) {

                e.printStackTrace();
                codigoRetorno = "Erro - Tente Novamente";
        }

        ps.close();
        rs.close();

        return codigoRetorno;
    }

    public String contabilDebito(Cliente cliente, Double valorCompra) throws
SQLException{

        Connection connection = null;
```

```
PreparedStatement ps = null;
ResultSet rs = null;

try {

        connection = Conexao.estabeleceConexao();

        StringBuilder query = new StringBuilder();

        query.append(" SELECT a.SD_ATUAL, b.BLOQUEADO,
b.DT_VENCIMENTO, a.NUM_CONTA");
        query.append(" FROM conta_corrente a INNER JOIN plastico b");
        query.append(" ON a.clientes_MCI = b.clientes_MCI");
        query.append(" WHERE a.clientes_MCI = ?");

        ps = connection.prepareStatement(query.toString());

        ps.setInt(1, cliente.getMciCliente());

        rs = ps.executeQuery();

        if(rs.next()){

                saldoAtual = rs.getDouble("SD_ATUAL");
                bloqueado = rs.getString("BLOQUEADO");
                dataVencimento = rs.getDate("DT_VENCIMENTO");
                nrContaCorrente = rs.getInt("NUM_CONTA");

                if( valorCompra > saldoAtual){

                        codigoRetorno = "Saldo Insuficiente";
                        return codigoRetorno;

                }

                if( bloqueado.equals("S")){

                        codigoRetorno = "Cartão Bloqueado";
                        return codigoRetorno;
                }

                if (dataVencimento.compareTo(new Date()) < 0 ){

                        codigoRetorno = "Cartão Vencido";
                        return codigoRetorno;

                }else {
```

```
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-
MM-dd");
                Date hoje = new Date();
                Calendar calendar = Calendar.getInstance();
                calendar.setTime(hoje);

                String dataTransacao = sdf.format(calendar.getTime());

                        try {

                                StringBuilder sqlGrava = new StringBuilder();

                                sqlGrava.append(" UPDATE conta_corrente SET
SD_ANTERIOR = ?,");
                                sqlGrava.append(" SD_ATUAL = ?,
VL_ULT_TRNS = ?, LCL_ULT_TRNS = ?,");
                                sqlGrava.append(" DT_ULT_TRNS = ?,
TP_ATNTC = ? WHERE clientes_MCI = ?");

                                ps =
connection.prepareStatement(sqlGrava.toString());

                                ps.setDouble(1, saldoAtual);
                                ps.setDouble(2, (saldoAtual - valorCompra));
                                ps.setDouble(3, valorCompra);
                                ps.setString(4, "Mercearia do Zé");
                                ps.setString(5, dataTransacao);
                                ps.setInt(6, 1);
                                ps.setInt(7, cliente.getMciCliente());

                                ps.executeUpdate();

                                try {

                                        StringBuilder sqlGravaRegistro = new
StringBuilder();

                                        sqlGravaRegistro.append(" INSERT
INTO reg_conta_corrente (VLR_TRNS, LCL_TRNS, DT_TRNS,");
                                sqlGravaRegistro.append(" TP_AUT,
conta_corrente_NUM_CONTA)");
                                sqlGravaRegistro.append(" values (?, ?, ?, ?, ?)");

                                        ps =
connection.prepareStatement(sqlGravaRegistro.toString());

                                        ps.setDouble(1, valorCompra);
                                        ps.setString(2, "Mercearia do Zé");
                                        ps.setString(3, dataTransacao);
```

```
                                        ps.setInt(4, 1);
                                        ps.setInt(5, nrContaCorrente);

                                        ps.executeUpdate();

                                        codigoRetorno = "Aprovado";

                                    } catch (SQLException e) {

                                        e.printStackTrace();
                                        codigoRetorno = "Erro - Tente
Novamente";
                                    }


                                }

                                catch (SQLException e) {

                                    e.printStackTrace();
                                    codigoRetorno = "Erro - Tente Novamente";
                                }

                            }

                        }else{

                            codigoRetorno = "Operação Incorreta";
                        }

                    } catch (SQLException e) {

                        e.printStackTrace();
                        codigoRetorno = "Erro - Tente Novamente";
                    }

                    ps.close();
                    rs.close();

                    return codigoRetorno;

                }

                public String bloqueiaCartão(Cliente cliente){

                    Connection connection = null;
                    PreparedStatement ps = null;
                    ResultSet rs = null;
                    int qntErros = 0;
```

```
            try {
                connection = Conexao.estabeleceConexao();
                String sql = null;
                sql = " SELECT QNT_ERROS FROM plastico WHERE clientes_MCI
= ?";
                ps = connection.prepareStatement(sql);
                ps.setInt(1, cliente.getMciCliente());
                rs = ps.executeQuery();
                if(rs.next()){
                    qntErros = rs.getInt("QNT_ERROS");
                    if(qntErros >= 2 ){
                        try{
                        connection = Conexao.estabeleceConexao();
                        String sqlUpdateBloq = null;
                        sqlUpdateBloq = " UPDATE plastico SET
BLOQUEADO = ?, QNT_ERROS = 3 WHERE clientes_MCI = ?";
                        ps = connection.prepareStatement(sqlUpdateBloq);
                        ps.setString(1, "S");
                        ps.setInt(2, cliente.getMciCliente());
                        ps.executeUpdate();
                        codigoRetorno = "Cartão Bloqueado";
                    }catch(Exception e){
                         e.printStackTrace();
                         codigoRetorno = "Erro - Tente Novamente";
                    }
                    }else{
              try{
```

```
                                        connection = Conexao.estabeleceConexao();

                                        String sqlUpdateErros = null;
                                        sqlUpdateErros = " UPDATE plastico SET
QNT_ERROS = ? WHERE clientes_MCI = ?";

                                        ps = connection.prepareStatement(sqlUpdateErros);
                                        ps.setInt(1, qntErros + 1);
                                        ps.setInt(2, cliente.getMciCliente());

                                        ps.executeUpdate();

                                        ps.close();

                                        codigoRetorno = "Digital Inválida";

                                    }catch(Exception e){

                                         e.printStackTrace();
                                         codigoRetorno = "Erro - Tente Novamente";

                                    }

                                    }

                            }

                    ps.close();
                    rs.close();

            }

            catch(SQLException e){

               e.printStackTrace();
               codigoRetorno = "Erro - Tente Novamente";

            }

            return codigoRetorno;

              }

}
```

Transcreve-se abaixo código da classe *Java* responsável pela geração dos limites:

```
package br.com.vinicius.action;

import br.com.vinicius.Cliente;

public class GeraLimites {

   public static float limiteCartao(Cliente cliente){

      float limiteCartao = 0.0f;

      String renda = cliente.getRendaCliente().toString();
      float rendaFinal = Float.parseFloat(renda.replace(",", "."));


      if("Empresário".equals(cliente.getNatOcupacaoCliente())){

            limiteCartao = (float) 0.8*rendaFinal;

      }else if ("Empregado Público".equals(cliente.getNatOcupacaoCliente())){

            limiteCartao = (float) 1.2*rendaFinal;

      }
      else if ("Empregao Privado".equals(cliente.getNatOcupacaoCliente())){

            limiteCartao = (float) 1.0*rendaFinal;

      }else if ("Sem Vínculo Emprego".equals(cliente.getNatOcupacaoCliente())){

            limiteCartao = (float) 0.4*rendaFinal;

      }

      return limiteCartao;

   }

   public static float limiteContaCorrente(Cliente cliente){

      float limiteContaCorrente = 0.0f;

      String renda = cliente.getRendaCliente().toString();
      float rendaFinal = Float.parseFloat(renda.replace(",", "."));


      if("Empresário".equals(cliente.getNatOcupacaoCliente())){
```

```
            limiteContaCorrente = (float) 0.6*rendaFinal;

        }else if ("Empregado Público".equals(cliente.getNatOcupacaoCliente())){

            limiteContaCorrente = (float) 0.8*rendaFinal;

        }
        else if ("Empregao Privado".equals(cliente.getNatOcupacaoCliente())){

            limiteContaCorrente = (float) 0.5*rendaFinal;

        }else if ("Sem Vínculo Emprego".equals(cliente.getNatOcupacaoCliente())){

            limiteContaCorrente = (float) 0.3*rendaFinal;

        }

        return limiteContaCorrente;

    }

}
```

Transcreve-se abaixo código da *Applet* inserida no cartão:

```
package cartao;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.JCSystem;
import javacard.framework.OwnerPIN;
import javacard.framework.Util;

public class GravaRecuperaPIN extends Applet {

            // codigo CLA byte no comando APDU header
            private final static byte CLA = (byte)0x33;

            // codigo INS byte no comando APDU header
            private final static byte RECUPERA_PIN = (byte)0x20;
            private final static byte ALTERAR_PIN = (byte) 0x60;
            private final static byte GRAVAR_MCI = (byte) 0x42;
            private final static byte RECUPERA_MCI = (byte) 0x52;
```

```
        //tentativas PIN
        private final static byte TENTATIVAS_PIN = (byte) 0x03;

        //tamanho maximo e minimo
        private final static byte TAMANHO_MAX_PIN = (byte) 99;
        private final static byte TAMANHO_MIN_PIN = (byte) 1;

        //codigo de respostas
        private final static short SW_PIN_INCORRETO = (short) 0x6300;
        private final static short SW_PIN_NECESSARIO = (short) 0x6301;
        private final static short SW_PIN_MUITO_GRANDE = (short) 0x6E86;
        private final static short SW_PIN_MUITO_PEQUENO = (short) 0x6E87;
        private final static short SW_PIN_BLOQUEADO = (short) 0x6E89;
        private final static short SW_PIN_EM_BRANCO = (short) 0x6E88;

        // Variaveis de escopo
        private byte[] dados = null;
        private byte[] mci = null;
        private short valorMci = null;
        private OwnerPIN ownerPin;
        private OwnerPIN ownerMci;
        private boolean inicioUPIN = false;


        // Metodo construtor da Classe
        public GravaRecuperaPIN() {
                ownerPin = new OwnerPIN(TENTATIVAS_PIN,
TAMANHO_MAX_PIN);
                register(); // Registra a instancia da applet e torna selecionavel
        }


        // Metodo que instala a Applet no Smart Card
        public static void install(byte[] bArray, short bOffset, byte bLength) {

                new GravaRecuperaPIN();
        }

        // Metodo Recuperar PIN
        private void recuperarPin(APDU apdu) {
                if(!inicioUPIN) {
                        ISOException.throwIt(SW_PIN_EM_BRANCO);
                        return;
                }
                if(ownerPin.getTriesRemaining()==0){
                        ISOException.throwIt(SW_PIN_BLOQUEADO);
                        return;
                }
                byte[] buffer = apdu.getBuffer(); //valor do PIN
```

```
                byte byteRead = (byte)(apdu.setIncomingAndReceive()); //tamanho do
PIN

                // informa ao JCRE que será enviado uma resposta
        apdu.setOutgoing();
                // informa ao JCRE o tamanho da mensagem em bytes
                apdu.setOutgoingLength((short)len_text);
                // envia a mensgem para o host
        apdu.sendBytesLong(pin_card, (short)0, (short)len_text); // Envio número de mci R-
APDU
        }

        // Metodo Alterar PIN
        private void alterarPin(APDU apdu) {

                byte[] buffer = apdu.getBuffer(); //valor do PIN
                byte numBytes = buffer[ISO7816.OFFSET_LC]; //tamanho do PIN


                if ( numBytes > TAMANHO_MAX_PIN ){
                        ISOException.throwIt(SW_PIN_MUITO_GRANDE);
                }
                if ( numBytes < TAMANHO_MIN_PIN ){
                        ISOException.throwIt(SW_PIN_MUITO_PEQUENO);
                }
                short offset_cdata = buffer[ISO7816.OFFSET_CDATA];
                ownerPin.update(buffer, offset_cdata, numBytes);
                ownerPin.resetAndUnblock();
                inicioUPIN = true;
        }

        //metodo Recuperar MCI
        public void recuperaMci(APDU apdu) {

                //verifica se existe PIN e MCI gravados
                if (inicioUPIN && !ownerMci.isValidated()){
                        ISOException.throwIt(SW_PIN_NECESSARIO);
                }

                //recupera dados da mensagem APDU recebida da classe
RecuperaPIN.java
                byte[] buffer = apdu.getBuffer();

                //tamanho do dado esperado
                short le = (short)(buffer[ISO7816.OFFSET_LC] & 0x00FF);

                //tamando do dado armazenado na variavel
                short len_text = (short)mci.length;
```

```
                        //verifica se o tamanho do dado esperado eh igual ao que ele recebera
                        if (le != len_text) {
                                ISOException.throwIt((short)
(ISO7816.SW_CORRECT_LENGTH_00 + len_text));
                        }
                        // informa ao JCRE que será enviado uma resposta
        apdu.setOutgoing();
                        // informa ao JCRE o tamanho da mensagem em bytes
                        apdu.setOutgoingLength((short)len_text);
                        // envia a mensgem para o host
        apdu.sendBytesLong(mci, (short)0, (short)len_text); // Envio número de mci R-APDU
                }

                public void gravaMci(APDU apdu){

                        mci = apdu.getBuffer(); //valor do MCI array de bytes
                        byte numBytes = mci[ISO7816.OFFSET_LC]; //tamanho do MCI

                        // Valor MCI campo Data
                        short valorMci = mci[ISO7816.OFFSET_CDATA];
                        ownerMci.update(buffer, valorMci, numBytes);
                        ownerMci.resetAndUnblock();
                }


                // Processa a mensagem APDU
                public void process(APDU apdu) throws ISOException {
                        // Captura o conteudo da mensagem APDU
                        byte[] buffer = apdu.getBuffer();

                        // Processa a mensagem APDU de acordo com o codigo da instrucao
(INS)
                        switch (buffer[ISO7816.OFFSET_INS]) {
                                case RECUPERA_PIN: autenticarPin(apdu); break;

                                case ALTERAR_PIN: alterarPin(apdu); break;
                                case RECUPERA_MCI: recuperaMci(apdu); break;
                                case GRAVAR_MCI: gravaMci(apdu); break;
                        }
                }
}
```

Transcreve-se abaixo o código da classe Java responsável por executar os arquivos .bat:

```
package br.com.vinicius.smartcard;

import java.io.IOException;

public class CompilaArquivoBat {

	private boolean controle = false;

	public boolean carregaArquivoBat(String arquivoBat){

		if(arquivoBat.equals("Compila_ClasseGravaRecuperaPIN")){
			try {

executarComando("C:\Users\user\Desktop\Desenvolvimento\Codigos\smartcard\Compila_ClasseGravaRecuperaPIN.bat");
			} catch (IOException e) {
				e.printStackTrace();
			}
			controle = true;
		}else if(arquivoBat.equals("Instalar_AppletSmartCard")){
			try {

executarComando("C:\Users\user\Desktop\Desenvolvimento\Codigos\smartcard\Instalar_AppletSmartCard.bat");
			} catch (IOException e) {
				e.printStackTrace();
			}
			controle = true;
		}else if(arquivoBat.equals("CompilaExecuta_RecuperaATR")){
			try {

executarComando("C:\Users\user\Desktop\Desenvolvimento\Codigos\smartcard\CompilaExecuta_RecuperaATR.bat");
			} catch (IOException e) {
				e.printStackTrace();
			}
			controle = true;
		}else if(arquivoBat.equals("CompilaExecuta_GravaPIN")){
			try {
```

```
executarComando("C:\Users\user\Desktop\Desenvolvimento\Codigos\smartcard\CompilaExec
uta_GravaPIN.bat");
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        controle = true;
                }else if(arquivoBat.equals("CompilaExecuta_RecuperaPIN")){
                        try {


executarComando("C:\Users\user\Desktop\Desenvolvimento\Codigos\smartcard\CompilaExec
uta_RecuperaPIN.bat");
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        controle = true;
                }
                return controle;
        }

        public void executarComando (String comando) throws IOException {

                Process proc = Runtime.getRuntime().exec(comando);

                try {
                //aguarda a aplicacao terminar
                        proc.waitFor();
                }
                catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

Transcreve-se abaixo o código da classe Java que captura o ATR do cartão e insere em banco de dados:

```
import javax.smartcardio.*;

import java.sql.*;
import java.util.List;
import java.util.ListIterator;

import com.sinergbrasil.jc.*;
```

```
//Classe reponsavel por capturar o ATR do Smart Card
public class RecuperaATR extends java.lang.Object {

	private static String nroATR;

	//construtor
	public RecuperaATR() {
	}

	public static void main(java.lang.String[] argv) {

		try {
			// Leitores de Smart Card
			TerminalFactory factory = TerminalFactory.getDefault();
			List terminalList = factory.terminals().list();

			// Seleciona leitor de Smart Card
			CardTerminal terminal = (CardTerminal) terminalList.get(0);

			// Aguarda o cartao ser inserido
			terminal.waitForCardPresent(10000);

			// Faz a conexao com o cartao. "*" para automatico
			Card card = terminal.connect("*");

			// Abertura do canal de comunicaco
			CardChannel ch = card.getBasicChannel();

			// Captura o ATR do cartao
			ATR atr = card.getATR();
			nroATR = arrayToHex(atr.getBytes());

			//recupera ultimo MCI cadastrado
			mci = recuperaUltimoMCI();

			//grava ATR no banco de dados passando
			gravaATR(nroATR, mci);

			card.disconnect(false);

		} catch (Exception e) {
			e.printStackTrace();
		}
	}


	public static int recuperaUltimoMCI() throws SQLException{
```

```
        int mciCliente = 0;
Connection con = null;
PreparedStatement ps = null;
        ResultSet rs = null;
try{
                con = Conexao.estabeleceConexao();
                String sqlQuery = null;
                sqlQuery = "SELECT MAX(MCI) FROM clientes";
                ps = con.prepareStatement(sqlQuery);
                rs = ps.executeQuery();
                if(rs.next()){
                        mciCliente = rs.getInt("MAX(MCI)");
                }
                ps.close();
                rs.close();
        }
        catch(SQLException e){
                e.printStackTrace();
        }
        Conexao.fecharConexao();
        return mciCliente;
  }

  public static void gravaATR(String nroATR, int mciCliente) throws SQLException {

         Connection con = null;
         PreparedStatement ps = null;
         ResultSet rs = null;
         try{
                 con = Conexao.estabeleceConexao();
                 String sql = null;
                 sql = "UPDATE plastico SET ATR = ? WHERE clientes_MCI = ?";
                 ps = con.prepareStatement(sql);
                 ps.setString(1, nroATR);
                 ps.setInt(2, mciCliente);
                 rs = ps.executeUpdate();
                 ps.close();
                 rs.close();
         }
         catch(SQLException e){
                 e.printStackTrace();
         }
         Conexao.fecharConexao();
         }


  // Converte array de bytes em uma string hexadecimal
  public static String arrayToHex(byte[] data) {
```

```
                StringBuffer sb = new StringBuffer();
                for(int i = 0; i < data.length; i++) {
                        String bs = Integer.toHexString(data[i] & 0xFF);
                        if(bs.length() == 1) {
                                sb.append(0);
                        }
                        sb.append(bs);
                }
                return sb.toString();
        }

}
```

Trancreve-se abaixo código da classe Java responsável por enviar a mensagem APDU que grava o PIN no cartão:

```
import javax.smartcardio.*;

import java.sql.*;
import java.util.List;
import java.util.ListIterator;

import com.sinergbrasil.jc.*;

//Classe responsavel por gravar o PIN do usuario no Smart Card
public class GravarPIN extends java.lang.Object {

        // APDU de Select
        private static CommandAPDU SELECT_APDU = new CommandAPDU(
                0x00, 0xa4, 0x04, 0x00,
                new byte[] {(byte)0xa0,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x62,
(byte)0x03,(byte)0x01,(byte)0x0d,(byte)0x01,(byte)0x02}
        );

        private static int mci;
        private static String pin_card;

        public GravarPIN() {
        }

        public static void main(java.lang.String[] argv) {

                try {
```

```
                        // Leitores de Smart Card
                        TerminalFactory factory = TerminalFactory.getDefault();
                        List terminalList = factory.terminals().list();

                        // Seleciona leitor de Smart Card
                        CardTerminal terminal = (CardTerminal) terminalList.get(0);

                        // Faz a conexao com o cartao. "*" para automatico
                        Card card = terminal.connect("*");

                        // Abertura do canal de comunicaco
                        CardChannel ch = card.getBasicChannel();

                        // Envia o APDU de SELECT
                        ResponseAPDU ra = ch.transmit(SELECT_APDU);

                        // Recupera MCI
                        mci = recuperaUltimoMCI();

                        //Hash
                        pin_card = md5(recuperaFormaPIN(mci));

                        //Gravar Senha
                        CommandAPDU MENSAGEM_APDU = new CommandAPDU(0x33,
0x60, 0x00, 0x00, pin_card.getBytes());

                        //Gravar MCI
                        CommandAPDU MENSAGEM_APDU_MCI = new
CommandAPDU(0x33, 0x42, 0x00, 0x00, mci.getBytes());

                        ra = ch.transmit(MENSAGEM_APDU);

                        card.disconnect(false);

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public String recuperaFormaPIN(int mci){

        String pin_card = null;
        Connection connection = null;
        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
                connection = Conexao.estabeleceConexao();
```

```
                StringBuilder query = new StringBuilder();
                query.append(" SELECT a.NR_CONTA_CARTAO, b.ATR");
                query.append(" FROM conta_cartao a INNER JOIN plastico b");
                query.append(" ON a.clientes_MCI = b.clientes_MCI");
                query.append(" WHERE a.clientes_MCI = ?");

                ps = connection.prepareStatement(query.toString());
                ps.setInt(1, mci);
                rs = ps.executeQuery();

                if(rs.next()){
                        if(rs.getInt("NR_CONTA_CARTAO") != 0){
                                pin_card = rs.getString("ATR") + mci.toString() +
rs.getInt("NR_CONTA_CARTAO") + 2;
                        }
                }
                else{
                        query = new StringBuilder();
                        query.append(" SELECT a.NUMERO, b.ATR");
                        query.append(" FROM conta_corrente a INNER JOIN plastico b");
                        query.append(" ON a.clientes_MCI = b.clientes_MCI");
                        query.append(" WHERE a.clientes_MCI = ?");

                        ps = connection.prepareStatement(query.toString());
                        ps.setInt(1, mci);
                        rs = ps.executeQuery();

                        if(rs.next()){
                                if(rs.getInt("NUMERO") != 0){
                                        pin_card = rs.getString("ATR") + mci.toString() +
rs.getInt("NUMERO") + 1;
                                }
                        }
                }
                ps.close();
                rs.close();
        }
        catch(SQLException e){
                e.printStackTrace();
        }
        Conexao.fecharConexao();

        return pin_card;

        }

        // Converte array de bytes em uma string hexadecimal
        public static String arrayToHex(byte[] data) {
                StringBuffer sb = new StringBuffer();
```

```
            for(int i = 0; i < data.length; i++) {
                String bs = Integer.toHexString(data[i] & 0xFF);
                if(bs.length() == 1) {
                    sb.append(0);
                }
                sb.append(bs);
            }
            return sb.toString();
        }

        //Função para criar hash do PIN
        public static String md5(String pin){

            String pinHash = null;
            MessageDigest md = null;

            try {
                md = MessageDigest.getInstance("MD5");
            }
            catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }

            BigInteger hash = new BigInteger(1, md.digest(pin.getBytes()));
            pinHash = hash.toString(16);

            return pinHash;
            }

}
```

Transcreve-se abaixo o código da classe Java responsável por recuperar PIN armazenado:

```
import javax.smartcardio.*;

import java.sql.*;
import java.util.List;
import java.util.ListIterator;

import com.sinergbrasil.jc.*;

//Classe responsavel por autenticar o PIN do usuario
public class RecuperaPIN  extends java.lang.Object  {
```

```
        // APDU de Select
        private static CommandAPDU SELECT_APDU = new CommandAPDU(
                0x00, 0xa4, 0x04, 0x00,
                new byte[] {(byte)0xa0,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0x62,
(byte)0x03,(byte)0x01,(byte)0x0d,(byte)0x01,(byte)0x02}
        );

        private static int mci;
        private static String pin_card;

        public RecuperaPIN() {
        }

        public static void main(java.lang.String[] argv) {

                try {
                        // Leitores de Smart Card
                        TerminalFactory factory = TerminalFactory.getDefault();
                        List terminalList = factory.terminals().list();

                        // Seleciona leitor de Smart Card
                        CardTerminal terminal = (CardTerminal) terminalList.get(0);

                        // Aguarda o cartao ser inserido
                        terminal.waitForCardPresent(10000);

                        // Faz a conexao com o cartao. "*" para automatico
                        Card card = terminal.connect("*");

                        // Abertura do canal de comunicacao
                        CardChannel ch = card.getBasicChannel();

                        // Envia o APDU de SELECT
                        ResponseAPDU ra = ch.transmit(SELECT_APDU);
                        ResponseAPDU ra2 = ch.transmit(SELECT_APDU);

                        //Recuperar PIN - campo de dados em branco 0x00
                        CommandAPDU MENSAGEM_APDU = new CommandAPDU(0x33,
0x20, 0x00, 0x00, 0x00);

                        //Recuperar MCI - campo de dados em branco 0x00
                        CommandAPDU MENSAGEM_APDU_MCI = new
CommandAPDU(0x33, 0x52, 0x00, 0x00, 0x00);

                        ra = ch.transmit(MENSAGEM_APDU);

                        //R-APDU trazendo os dados do MCI
                        ra2 = ch.transmit(MENSAGEM_APDU_MCI);
```

```
                    //converte array de bytes em hexadecimal
                    String temp = arrayToHex(ra2.getData());

                    //converte hexadecimal em ASCII
                    mci = Integer.parseInt(convertHexToString(temp));

                    gravaMciTransacao(mci);

                    card.disconnect(false);

            } catch (Exception e) {
                    e.printStackTrace();
            }
    }

    /*converte hexadecimal para ASCII – retirado de
http://www.mkyong.com/java/howto-convert-hex-to-ascii-in-java*/
    public String convertHexToString(String hex){

            StringBuilder sb = new StringBuilder();
            StringBuilder temp = new StringBuilder();

            for( int i=0; i < hex.length()-1; i+=2 ){
                    //pega o hexa em pares
                    String output = hex.substring(i, (i + 2));
                    //converte hexa para decimal
                    int decimal = Integer.parseInt(output, 16);
                    //converte decimal para caracter
                    sb.append((char)decimal);
                    temp.append(decimal);
            }
            return sb.toString();
    }

    // Converte array de bytes em uma string hexadecimal
    public static String arrayToHex(byte[] data) {

            StringBuffer sb = new StringBuffer();
            for(int i = 0; i < data.length; i++) {
                    String bs = Integer.toHexString(data[i] & 0xFF);
                    if(bs.length() == 1) {
                            sb.append(0);
                    }
                    sb.append(bs);
            }
            return sb.toString();
    }

    public static void gravaMciTransacao(int mci) throws SQLException{
```

```
        Connection con = null;
        PreparedStatement ps = null;

        try{

                con = Conexao.estabeleceConexao();

                String sql = null;
                sql = " UPDATE mci_transacao SET MCI = ?";

                ps = con.prepareStatement(sql);
                ps.setInt(1, mci);

                ps.executeUpdate();

                ps.close();

        }catch(Exception e){

                 e.printStackTrace();

        }

    }


}
```

Transcreve-se abaixo código da classe *Java* responsável pelas conexões com banco de dados:

```
        package br.com.vinicius;

import java.sql.Connection;
import java.sql.DriverManager;

public class Conexao {

    public static void main(String[] args) {

      estabeleceConexao();

    }
```

```
    private static final String DRIVER = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql://127.0.0.1:3306/projeto_final";
    private static final String DB_USER = "root";
    private static final String DB_PWD = "1205vini";

    public Conexao(){

    }

    public static Connection conexao;

    public static Connection estabeleceConexao(){

        try{

            Class.forName(DRIVER).newInstance();
            conexao = DriverManager.getConnection(URL, DB_USER, DB_PWD);

        }

        catch (Exception e) {

            try{

                Class.forName(DRIVER).newInstance();
                conexao = DriverManager.getConnection(URL, DB_USER, DB_PWD);

            }

            catch(Exception exception){

                  e.printStackTrace();
            }

        }
            return conexao;

    }

    public static void fecharConexao(){

            conexao = null;

    }

}
```

Transcreve-se abaixo *script* do banco de dados:

```
CREATE TABLE  `projeto_final`.`clientes` (
  `MCI` int(10) NOT NULL,
  `CPF` int(20) NOT NULL,
  `IDENTIDADE` varchar(20) NOT NULL,
  `NR_CONTA` int(20) NOT NULL,
  `NM_TITULAR` varchar(255) NOT NULL,
  `NM_MAE` varchar(255) NOT NULL,
  `NM_PAI` varchar(255) NOT NULL,
  `OCUPACAO` varchar(100) NOT NULL,
  `NT_OCUPACAO` varchar(100) NOT NULL,
  `RENDA_TITULAR` varchar(100) NOT NULL,
  `TEL` varchar(20) NOT NULL,
  `RESIDENCIA` text NOT NULL,
  `SENHA` int(11) DEFAULT '0',
  `FINGER_PRINT` text DEFAULT NULL,
  `DT_INICIO` date DEFAULT NULL,
  `DT_FIM` date DEFAULT NULL,
  PRIMARY KEY (`MCI`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE  `projeto_final`.`conta_cartao` (
  `NR_CONTA_CARTAO` int(20) unsigned NOT NULL AUTO_INCREMENT,
  `BANDEIRA` varchar(20) NOT NULL,
  `LMT_ANTERIOR` varchar(45) DEFAULT NULL,
  `LMT_ATUAL` varchar(45) NOT NULL,
  `VLR_ULT_TRNS` varchar(45) DEFAULT NULL,
  `DT_ULT_TRNS` date DEFAULT NULL,
  `LCL_ULT_TRNS` varchar(100) DEFAULT NULL,
  `TP_ATNTC` char(1) DEFAULT NULL,
  `DT_INICIO` date NOT NULL,
  `DT_FIM` date NOT NULL,
  `clientes_MCI` int(20) NOT NULL,
  PRIMARY KEY (`NR_CONTA_CARTAO`),
  KEY `clientes_MCI` (`clientes_MCI`),
  CONSTRAINT `clientes_MCI` FOREIGN KEY (`clientes_MCI`) REFERENCES `clientes`
(`MCI`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;



CREATE TABLE  `projeto_final`.`conta_corrente` (
  `NUM_CONTA` int(20) unsigned NOT NULL AUTO_INCREMENT,
  `clientes_MCI` int(20) NOT NULL,
  `AGENCIA` varchar(45) NOT NULL,
  `SD_ANTERIOR` varchar(45) DEFAULT NULL,
  `SD_ATUAL` varchar(45) NOT NULL,
```

```
  `VL_ULT_TRNS` varchar(45) DEFAULT NULL,
  `DT_ULT_TRNS` date DEFAULT NULL,
  `LCL_ULT_TRNS` varchar(45) DEFAULT NULL,
  `TP_ATNTC` char(1) DEFAULT NULL,
  `DT_INICIO` date NOT NULL,
  `DT_FIM` date NOT NULL,
  PRIMARY KEY (`NUM_CONTA`)
) ENGINE=InnoDB AUTO_INCREMENT=10008 DEFAULT CHARSET=latin1;


CREATE TABLE  `projeto_final`.`plastico` (
  `NR_PLASTICO` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `conta_cartao_NR_CONTA_CARTAO` int(20) unsigned NOT NULL,
  `clientes_MCI` int(20) unsigned NOT NULL,
  `conta_NUM_CONTA` int(20) unsigned NOT NULL,
  `DT_VENCIMENTO` date NOT NULL,
  `NM_TITULAR` varchar(45) NOT NULL,
  `BANDEIRA` varchar(45) NOT NULL,
  `BLOQUEADO` char(1) NOT NULL,
  `DT_GERACAO` date NOT NULL,
  `QNT_ERROS` bigint(20) unsigned NOT NULL,
  `ATR` varchar(255) NOT NULL,
  PRIMARY KEY (`NR_PLASTICO`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;


CREATE TABLE  `projeto_final`.`reg_conta_cartao` (
  `VLR_TRNS` double NOT NULL,
  `LCL_TRNS` varchar(255) NOT NULL,
  `DT_TRNS` date NOT NULL,
  `TP_AUT` int(20) unsigned NOT NULL,
  `conta_cartao_NR_CONTA_CARTAO` int(20) unsigned NOT NULL,
  `ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`ID`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;


CREATE TABLE  `projeto_final`.`reg_conta_corrente` (
  `VLR_TRNS` double NOT NULL,
  `LCL_TRNS` varchar(255) NOT NULL,
  `DT_TRNS` date NOT NULL,
  `TP_AUT` int(20) unsigned NOT NULL,
  `conta_corrente_NUM_CONTA` int(20) unsigned NOT NULL,
  `ID` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`ID`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1;
```

ANEXOS

Transcreve-se abaixo o código de apoio para geração do arquivo CAP:

```
-out EXP JCA CAP
-exportpath .
-applet  0x01:0x02:0x03:0x04:0x05:0x06:0x07:0x08:0x09:0x00:0x00
smartcard.GravaRecuperaPIN
smartcard
0x01:0x02:0x03:0x04:0x05:0x06:0x07:0x08:0x09:0x00 1.0
```

Transcreve-se abaixo o código de apoio para instalação da *Applet* no *smart card*:

```
mode_211
enable_trace
enable_timer
establish_context
card_connect
select -AID a000000003000000
open_sc -security 1 -keyind 0 -keyver 0 -mac_key 404142434445464748494a4b4c4d4e4f
-enc_key 404142434445464748494a4b4c4d4e4f // Open secure channel
delete -AID 0102030405060708090000
delete -AID 0102030405060708090
install -file GravaRecuperarPIN.cap -nvDataLimit 2000 -instParam 00 -priv 2
get_status -element 10
get_status -element 20
get_status -element 40
card_disconnect
release_context
```