Estudo de Caso: Caixa Automático

Programação OO
Abril de 2016
Instituto de Computação – UNICAMP
Profa. Cecília Mary Fischer Rubira

1. Descrição do Problema

Nosso objetivo é desenvolver uma aplicação para um banco que mantém uma rede de caixas automáticos, onde os clientes podem consultar seus saldos e efetuar saques.

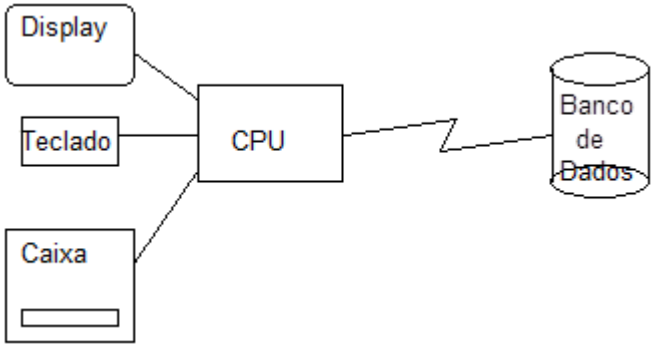


Fig. 1 - Arquitetura Física do Sistema de Caixa Automático

A Figura 1 apresenta as principais partes do sistema de caixa automático. Os caixas são operados através de um teclado numérico e um pequeno display, que substitui o vídeo. Toda a operação é controlada por uma CPU local, onde será instalada a aplicação. Há ainda uma ligação com o banco de dados que armazena as informações sobre as contas correntes e com o caixa propriamente dito, que fornece as cédulas aos clientes.

- Os caixas automáticos operam em dois modos distintos: supervisor e cliente.
- No modo supervisor, que é o modo inicial de operação, só é possível realizar uma operação de recarga de cédulas, efetuada por um supervisor que possua uma senha apropriada.
- Após uma operação de recarga, o caixa passa a ter R$1.000,00 em notas de R$10,00.
- No modo cliente, o caixa pode efetuar consultas de saldo e saques, exigindo a identificação do cliente através do número da conta e senha correspondente.
- Só são permitidos saques de valores múltiplos de R$10,00, até um máximo de R$200,00 e desde que haja saldo suficiente na conta do cliente.
- Na Figura 2 estão apresentados os vários casos de uso previstos para o sistema, usando a notação UML para diagramas de casos de uso (*use case diagrams*).

Diagrama de casos de uso

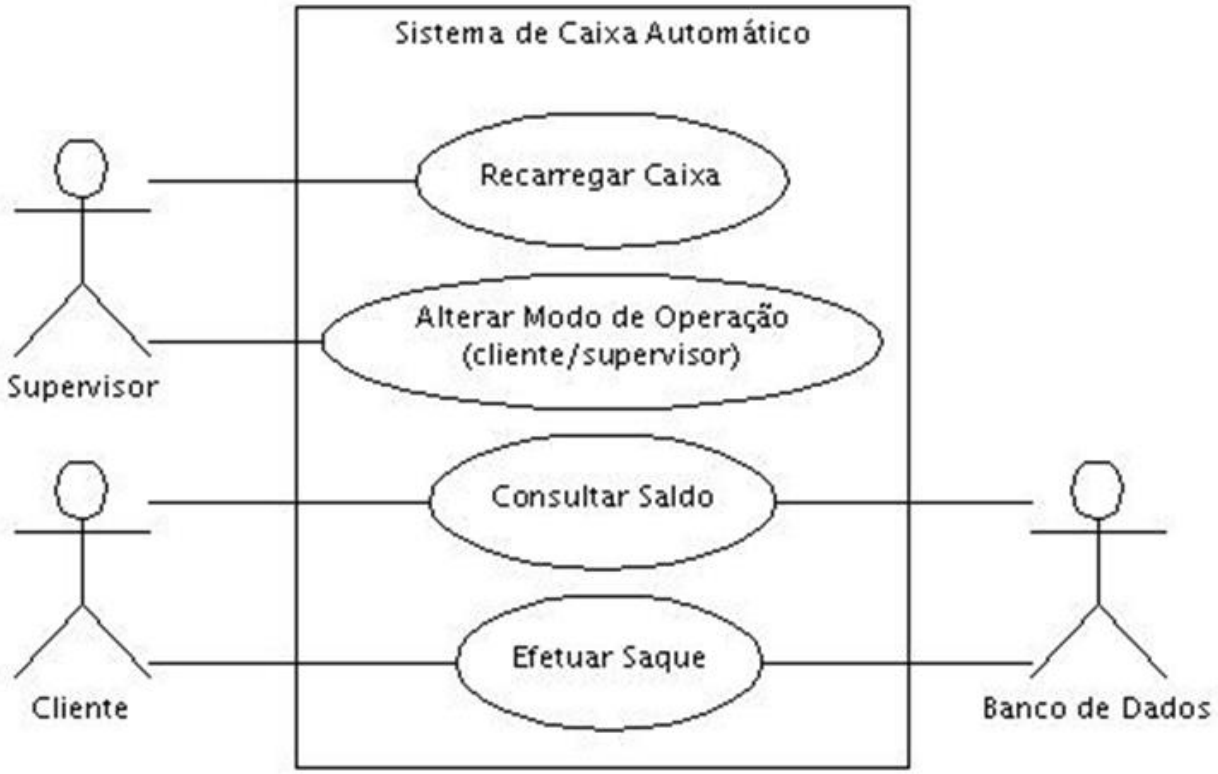
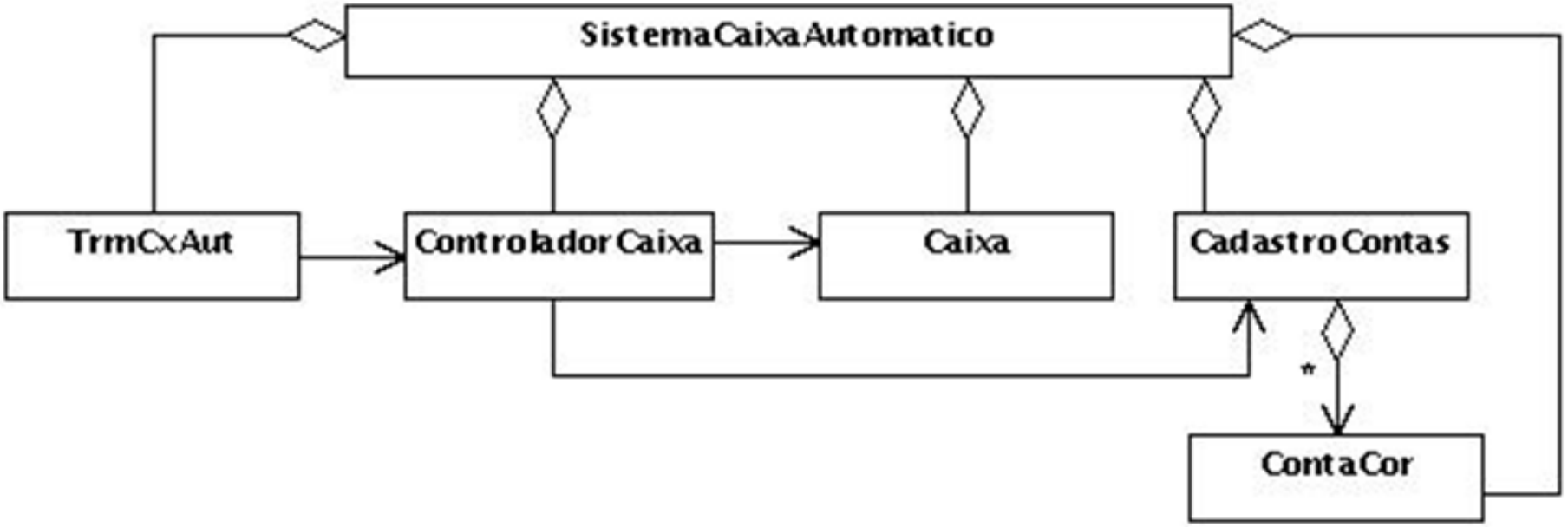


Diagrama de Classes



Análise e projeto do sistema

Classe **TrmCxAut**

- Um objeto dessa classe representa o terminal de um caixa automático, composto pelo teclado, display e caixa.
- Essa classe encapsula toda a interface com os usuários. Com isso, a aplicação pode ser adaptada para outros tipos de terminais que usem cartões magnéticos ou telas sensíveis ao toque, por exemplo, substituindo-se apenas essa classe, de forma transparente para o restante da aplicação.
- Como há uma CPU em cada caixa automático, haverá um único objeto desse tipo em cada instância do programa.

Classe **ControladorCaixa**

- Um objeto dessa classe encapsula a política do banco em relação aos saques em caixas automáticos e as interfaces entre um objeto da classe TrmCxAut com o restante do sistema.
- Um objeto controlador recebe as requisições do terminal e coordena a execução das demais classes do sistema. Existe apenas um único objeto controlador por caixa.

Classe **Caixa**

- Um objeto dessa classe representa um caixa propriamente dito, que armazena as cédulas para pagamento dos saques efetuados pelos clientes. O sistema de caixa automático contém apenas um objeto da classe Caixa, controlado pelo objeto ControladorCaixa.

Classe **ContaCor**

- Os objetos dessa classe são as contas mantidas pelos clientes do banco. Essa classe encapsula as políticas do banco para manutenção das contas correntes.

Classe **CadastroContas**

- Essa classe possui apenas uma instância, que representa o banco de dados que armazena as informações sobre as contas correntes.

2.1 Interface Pública da Classe TrmCxAut

- Um objeto da classe TrmCxAut possui duas operações: uma para executar um ciclo de operações do caixa e outra que permite mudar o seu modo de operação.
- A primeira operação é especificada por iniciarOperacao(), que inicializa o terminal e apresenta o menu de operações na tela.
- A segunda operação permite alternar entre os modos de operação “cliente” e “supervisor” (alternarModo(...)).
- Essa última operação é ativada por uma mensagem com um único parâmetro, do tipo inteiro, que informa a senha do supervisor, o único capaz de alterar o modo de operação de um caixa. Ambos métodos não retornam nenhum resultado.

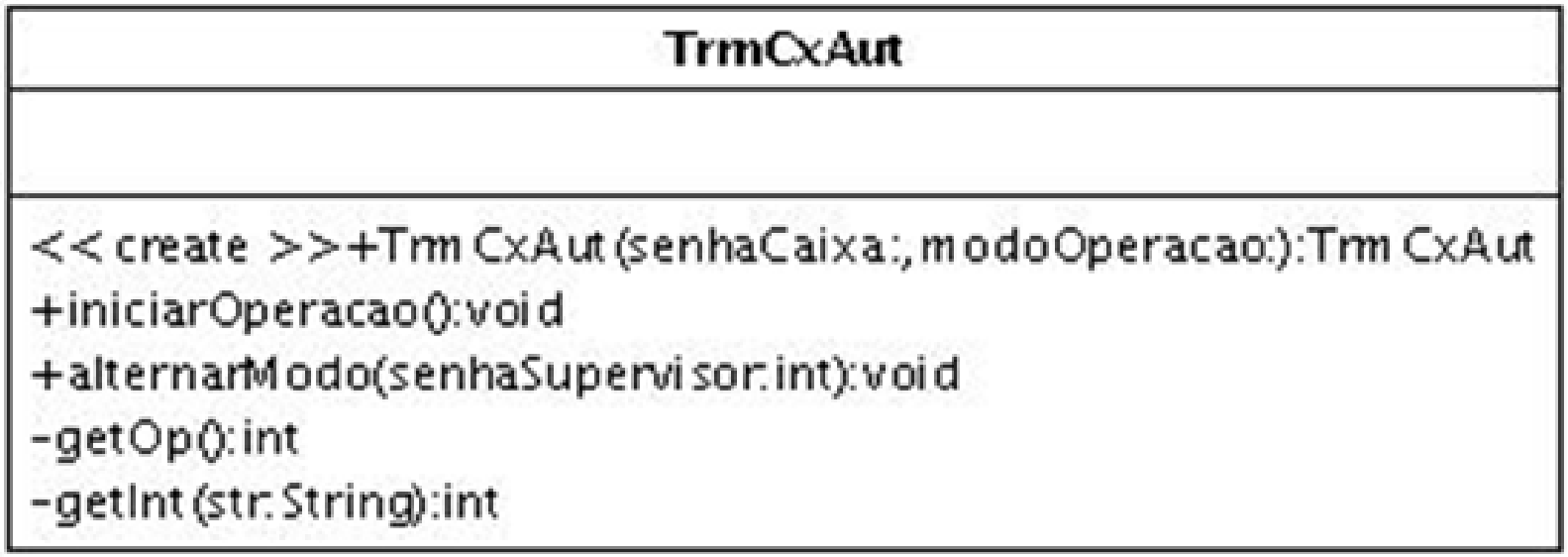


Figura 4 – Classe TrmCxAut

2.2 Interface Pública da Classe ControladorCaixa

- Um objeto da classe ControladorCaixa possui operações para controlar consultas de saldo, saques e recarregas do caixa, além da verificação da senha do supervisor.

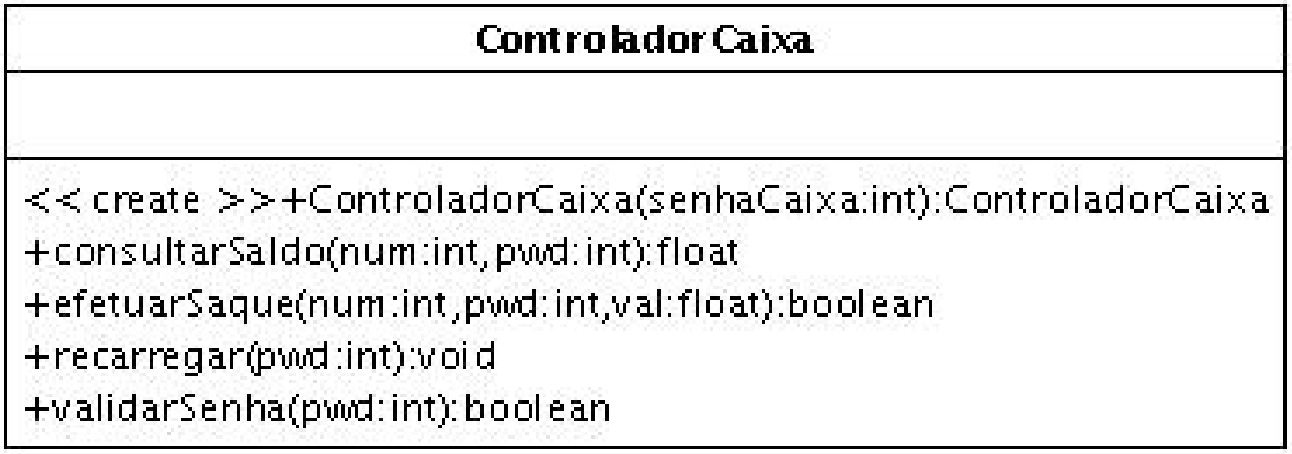
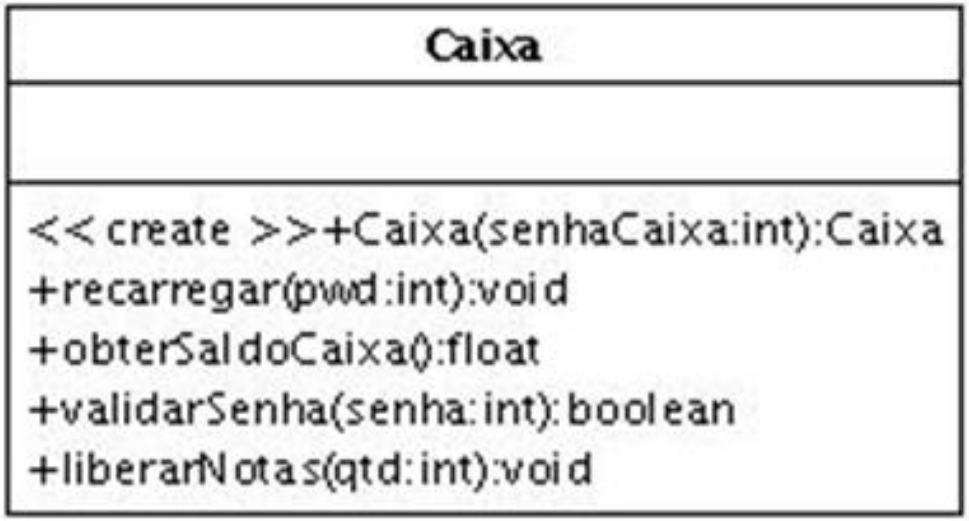


Figura 5 – Classe ControladorCaixa

- A operação consultarSaldo(...) deve retornar o valor do saldo da conta se o número da conta e a senha estiverem corretas, ou -1 em caso contrário.
- A operação efetuarSaque(...) deve retornar true, se o pedido de saque foi atendido, ou false, caso contrário.
- A operação recarregar(...) aciona a recarga do caixa automático.
- A operação validarSenha(...) solicita a verificação da senha do supervisor e retorna true, se a senha do supervisor estiver correta, ou false, caso contrário.

2.3 Interface Pública da Classe Caixa

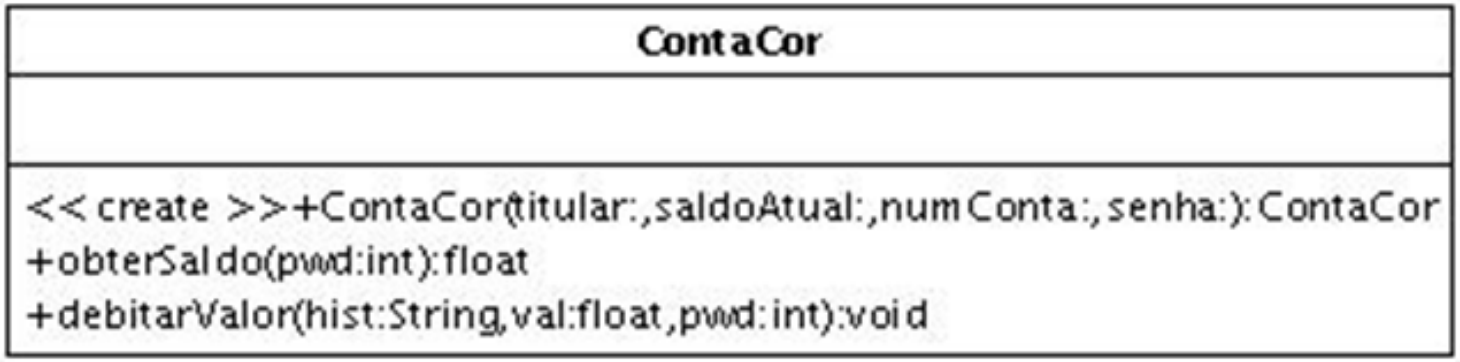
- Um objeto da classe Caixa possui operações para efetuar uma operação de recarga, consultar o saldo do caixa e validar a senha do supervisor.



- A operação recarregar(...) define o saldo do caixa como R$ 1000,00.
- A operação obterSaldoCaixa() deve retornar o valor do saldo do caixa eletrônico.
- A operação validarSenha(...) deve retornar true, se a senha informada for a senha do supervisor, ou false, caso contrário.
- A operação liberarNotas() controla o dispositivo que efetua o pagamento.

2.4 Interface Pública da Classe ContaCor

- Um objeto da classe ContaCor possui operações para obter o saldo da conta e debitar valores da mesma.



2.5 Interface Pública da Classe CadastroContas

- Para nossa aplicação a única operação relevante nessa classe é recuperar a referência de uma conta já existente, a partir do número da conta.

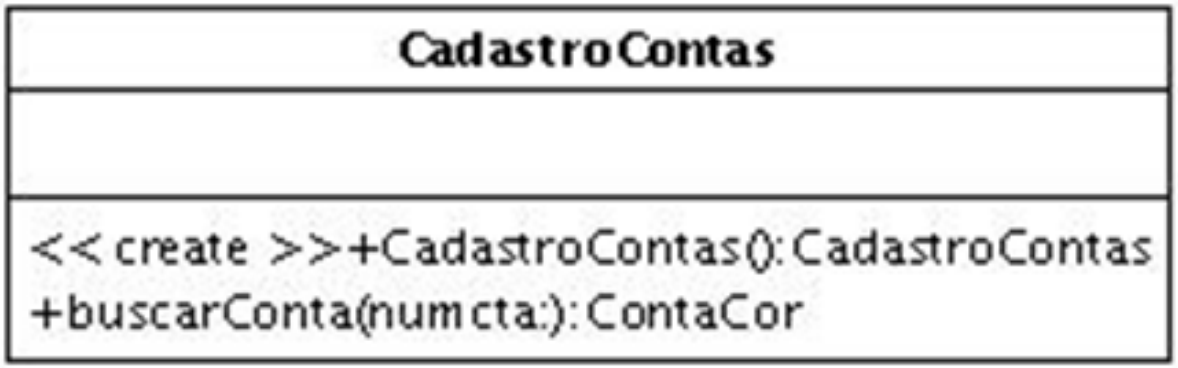


Figura 8 – Classe CadastroContas

2.6 Diagramas de Seqüência em UML

- As figuras 9, 10, 11 e 12 representam, através de diagramas de seqüência, as interações entre os objetos das classes da aplicação para realizar as ações definidas no diagrama de casos de uso.

Fig. 9 - Diagrama de Seqüência para Recarga do Caixa

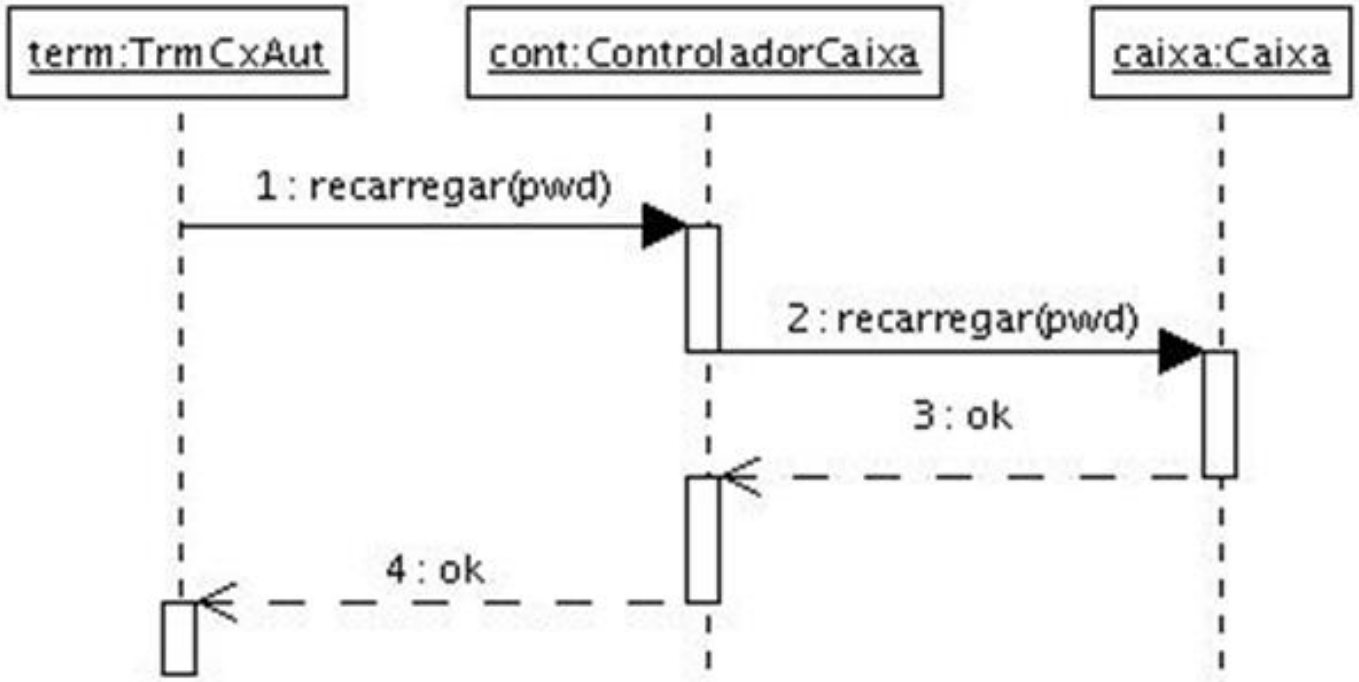


Fig. 10 - Diagrama de Seqüência para Alteração o Modo de Operação

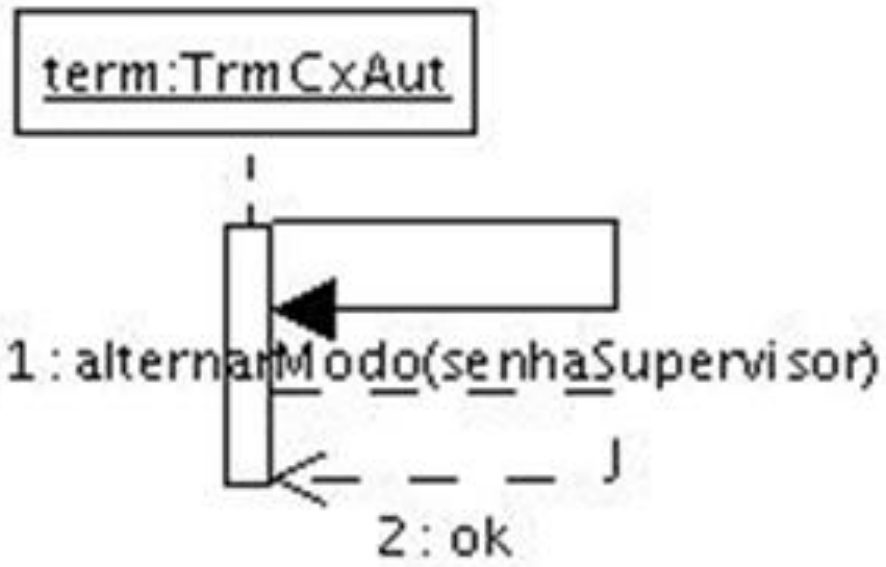


Fig. 11 - Diagrama de Seqüência para Consulta de Saldo

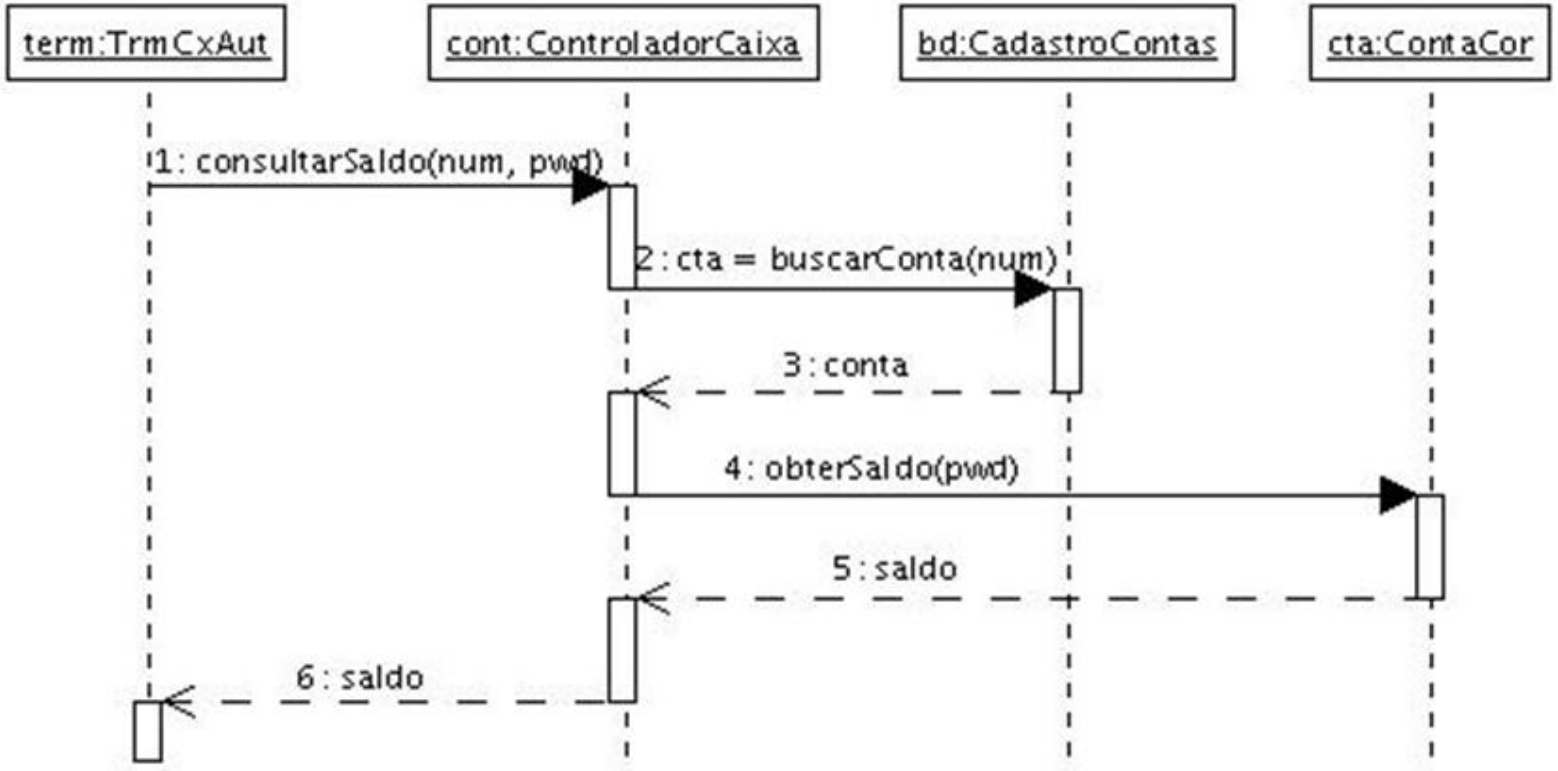
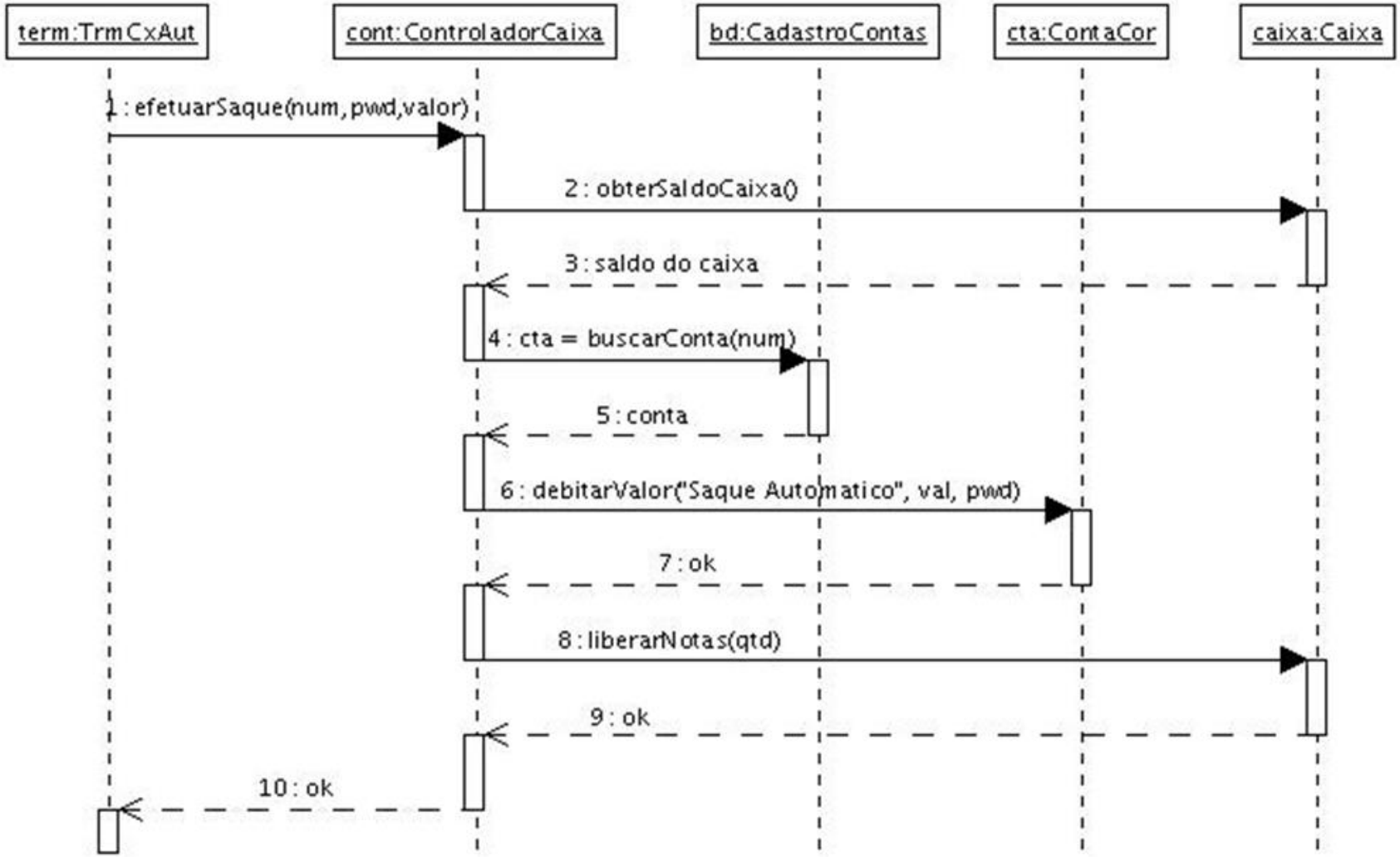


Fig. 12 - Diagrama de Seqüência para Saque



3. Implementação do Sistema

- Para cada classe da aplicação iremos criar um arquivo com sua especificação. Os arquivos devem ser gravados em formato texto, sem qualquer formatação, com o mesmo nome da classe e extensão .java

3.1 Especificação da Classe TrmCxAut

```
public class TrmCxAut {
          //Constantes
          // constantes que representam o modo de operação do terminal
          // (cliente ou supervisor)

          //Atributos
          //definição do estado da classe
          //Operacoes
          public TrmCxAut(int senhaCaixa, int modoOperacao) {
                    //código do construtor
          }

          public void iniciarOperacao(){
                    //código do método  iniciarOperacao()
          }

          public void alternarModo(int senhaSupervisor){
                    //código do método alternarModo()

          }

          private int getOp() {
                    //código do método getOp()
          }


          private int getInt(String str) {
                    //código do método getInt()
          }

}
```

Foram definidas duas constantes para representar os possíveis modos de operação do caixa automático: MODO_SUPERVISOR, equivalente ao valor inteiro 0 e MODO_CLIENTE, equivalente ao valor inteiro 1.

```
public static int MODO_SUPERVISOR = 0;
public static int MODO_CLIENTE = 1;
```

O estado de um objeto dessa classe pode ser descrito através de dois atributos: o controlador do caixa ao qual o terminal está acoplado e o modo de operação atual. Para isso definiremos uma variável de nome controladorCaixa, que irá conter uma referência para um objeto da classe ControladorCaixa, e outra de nome modoAtual, do tipo inteiro, para representar o modo de operação, podendo ser 0, para modo supervisor, ou 1, para modo cliente, conforme abaixo:

```
private ControladorCaixa controladorCaixa; // caixa que
processa as transações
private int modoAtual;  // modo de operação atual:
0=supervisor, 1=cliente
```

O método alternarModo(int senhaSupervisor)

A única ação executada por esse método é a atribuição de um novo valor à variável modoAtual, verificando a corretude da senha de administrador fornecida. O código seguinte implementa essa ação de forma a garantir a consistência do atributo, rejeitando atribuições inválidas.

```
if(this.controladorCaixa.validarSenha(senhaSupervisor)){
   if (this.modoAtual == TrmCxAut.MODO_SUPERVISOR)
      this.modoAtual = TrmCxAut.MODO_CLIENTE;
   else
      this.modoAtual = TrmCxAut.MODO_SUPERVISOR;
}
```

O método iniciarOperacao()

- Nesse método deve ser executado um ciclo de diversas operações, que podem ser consultas de saldo, saques, recargas do caixa e alteração do modo de operação.
- No modo cliente só podem ser executadas operações dos dois primeiros tipos, enquanto no modo supervisor somente operações de recarga.
- A operação para alterar o modo de execução está disponível tanto para clientes, quanto para supervisores, mas exige a digitação da respectiva senha de supervisão.
- Iremos incluir uma opção para encerrar um ciclo de operações que poderá ser selecionada a qualquer momento, apenas para facilitar os testes da aplicação.
- Podemos definir a seguinte estrutura geral, que utiliza um método auxiliar getOp, a ser definido mais adiante, para obter o código da operação solicitada pelo usuário, que poderá ser:
- 1 para consulta de saldo, 2 para saque, 3 para recarga do caixa, 8 para alterar o modeo de execução e 9 para encerrar o ciclo de operações.

```
int op;       // código da operação solicitada
              op=getOp();
              while (op!=9) {
                switch (op) {
                  case 1:
                   //código para solicitar o saldo
                  break;
                  case 2:
                     //código para solicitar o saque
                  break;
                  case 3:
                     //código para solicitar a recarga do
caixa
                  break;

                  case 8:
                     //código para alterar o modo de
operação do caixa
                  break;
                }
                op=getOp();
              }
```

- Conforme especificado no projeto, uma consulta de saldo é feita através da operação consultarSaldo(...), do objeto controladorCaixa, fornecendo o número da conta e a senha, ambos números inteiros. Um resultado igual a -1 indica que os dados fornecidos não são válidos. Em qualquer caso deve ser enviada uma resposta para o usuário com o valor do saldo ou uma mensagem de erro.
- O código seguinte implementa essa operação utilizando-se um método auxiliar, chamado getInt, para obter um número inteiro fornecido pelo usuário, tendo como parâmetro uma string, que será exigida ao usuário:

```
float saldo=controladorCaixa.consultarSaldo
                              (getInt("número da conta"),
getInt("senha"));
if (saldo==-1)  // testa se consulta foi rejeitada
   System.out.println("conta/senha inválida");
else System.out.println("Saldo atual: "+saldo);
```

Um pedido de saque pode ser implementado de forma semelhante, através da operação efetuarSaque(...), fornecendo-se, além do número da conta e senha, o valor solicitado pelo cliente:

```
boolean b=controladorCaixa.efetuarSaque
          (getInt("número da conta"), getInt("senha"),
getInt("valor"));
if (b)          // testa se saque foi aceito
   System.out.println("Pode retirar o dinheiro");
else System.out.println("Pedido de saque recusado");
```

Uma operação de recarga é ainda mais simples:

```
controladorCaixa.recarregar(getInt("senha"));
```

A operação de alteração do modo de operação é igualmente simples:

```
this.alternarModo(getInt("senha do supervisor"));
```

Iremos definir agora os métodos auxiliares getOp e getInt.

O método getOp()

Esse método deve obter o código da operação solicitada pelo usuário, garantindo que seja compatível com o modo de operação atual. O código seguinte implementa essa ação, utilizando o método getInt para obter a opção do usuário:

```
private int getOp() {
  int op;
  do {
    if (modoAtual==1) {  // modo cliente
      op=getInt
        ("opcao: 1 = consulta saldo, 2 = saque, 8=modo supervisor,
9=sai");
        if (op!=1 && op!=2 && op!=8 && op!=9) op=0;
    }else {                          // modo supervisor
        op=getInt
          ("opcao: 3 = recarrega, 8=modo cliente, 9=sai");
        if (op!=3 && op!=8 && op!=9) op=0;
    }
  } while (op==0);
  return(op);
}
```

O método getInt(String str)

A única ação desse método é obter um valor inteiro fornecido pelo usuário, após enviar uma mensagem solicitando o dado.

Para ler valores do teclado utilizando as classes das bibliotecas de Java precisamos definir mais duas variáveis que serão denominados r e st. A primeira é uma referência para um objeto de tipo Reader, associado à entrada padrão System.in, definida através do comando:

```
Reader r=new BufferedReader
        (new InputStreamReader (System.in));
```

A segunda variável é uma referência para um objeto da classe StreamTokenizer que faz o reconhecimento dos valores digitados pelo usuário, e é definida por:

```
StreamTokenizer st=new StreamTokenizer(r);
```

```
private int getInt(String str) {
  System.out.println("Entre com "+str);
  try {st.nextToken();}
  catch (IOException e) {
    System.out.println("Erro na leitura do teclado");
    return(0);
  }
  return((int)st.nval);
}
```

Como estão sendo utilizadas as classes Reader e StreamTokenizer, é necessário acrescentar, no início do arquivo e antes do comando class, a seguinte declaração, que especifica onde se encontram as definições dessas classes:

```
import
java.io.*;
```

O construtor da classe TrmCxAut

Um objeto da classe TrmCxAut contém um objeto da classe ControladorCaixa. O construtor da classe TrmCxAut é, portanto, o método mais indicado para criar esse objeto. Como visto anteriormente, utilizaremos a variável controladorCaixa para armazenar sua referência.

O código seguinte define um construtor para isso:

```
controladorCaixa = new ControladorCaixa(senhaCaixa);
modoAtual = modoOperacao;
```

Compilando a classe TrmCxAut

Após criado o arquivo TrmCxAut.java contendo toda a definição da classe podemos compilá-lo usando o comando:

```
javac TrmCxAut.java
```

Como a classe TrmCxAut contém referências para a classe CadastroContas, que ainda não existe, serão geradas diversas mensagens de erro de compilação, tal como ocorreu ao compilarmos a classe Caixa.

3.2 Especificação da Classe ControladorCaixa

```
public class ControladorCaixa {
        //Atributos
       //atributos da classe  ControladorCaixa

        //Operacoes

        public ControladorCaixa(int senhaCaixa) {
                //implementação do método construtor
        }


        public float consultarSaldo (int num, int pwd){
                //implementação do método consultarSaldo
        }

        public boolean efetuarSaque (int num, int pwd, float val){
                //implementação do método efetuarSaque
        }


        public void recarregar(int pwd){
                //implementação do método recarregar
        }

        public boolean validarSenha(int pwd){
                //implementação do método validarSenha
        }
}
```

O estado do controlador de um caixa pode ser definido através de dois atributos: o banco de dados de contas a ser utilizado e o próprio caixa automático. O primeiro atributo é do tipo CadastroContas, definido como:

```
private CadastroContas dbContas;  // Banco de dados das contas
```

O caixa automático é representado por um objeto da classe Caixa, definido como:

```
private Caixa caixa; // caixa automático
```

O método recarregar(int pwd)
Implemente este método, que requisita ao caixa a sua recarga.

O método validarSenha(int pwd)
Implemente este método, que requisita ao caixa a validação da senha de administração.

O método consultarSaldo(int num, int pwd)

- Para realizar uma operação de consulta de saldo, em consultarSaldo, é necessária a colaboração do objeto da classe ContaCor que representa a conta que está sendo consultada.
- O primeiro passo, portanto, será obter uma referência para esse objeto, a partir do número da conta recebido através do parâmetro num.
- Conforme especificado no projeto, será utilizado o método buscarConta do banco de dados.
- Devemos definir, portanto, uma variável auxiliar cta, de tipo ContaCor, que receberá o resultado do método buscarConta, passando como argumento o valor do parâmetro num, conforme abaixo:

```
ContaCor cta;
                cta = dbContas.buscarConta(num); // obtem
referencia para o objeto que representa a conta 'num'
```

Conforme especificado no projeto, caso o número informado não seja um número de conta válido, é retornada uma referência nula (valor null). Nesse caso o método consultarSaldo deve retornar o valor -1. Caso contrário devemos simplesmente repassar a consulta para o objeto que representa a conta e retornar o resultado obtido.
Podemos, portanto, concluir esse método com:

```
if (cta==null)   // se numero de conta invalido ...
   return -1; // ... retorna -1
else             // caso contrario ...
   return cta.obterSaldo(pwd); // efetua consulta
```

O método efetuarSaque(int num, int pwd, float val)

Implemente a operação de saque, que deve seguir a especificação:

1. Aceitar apenas valores múltiplos de R$ 10,00;
2. Saques de no máximo R$ 200,00;
3. Ter certeza que o caixa tem o dinheiro que o cliente deseja sacar;
4. Verificar se o saldo é válido;

No final, o método deve retornar true, caso o saque tenha sido efetuado com sucesso e false, caso contrário.

O construtor da classe

A implementação do construtor consistiu em instanciar tanto o caixa automático, quanto o banco de dados de contas que será utilizado. A solução adotada é apresentada a seguir:

```
this.senhaCaixa =  senhaCaixa;
dbContas = new CadastroContas();
caixa = new Caixa(senhaCaixa);
```

Compilando a classe ControladorCaixa

Após criado o arquivo ControladorCaixa.java contendo toda a definição da classe podemos compilá-la usando o comando:

```
javac ControladorCaixa.java
```

3.3 Especificação da Classe Caixa

```
public class Caixa {
        //Atributos
         //aqui entram os atributos

        //Operacoes
        public Caixa(int senhaCaixa) {
                //implementação do construtor
        }


        public void recarregar(int pwd){
                //implementação do método recarregar()
        }

        void liberarNotas(int qtd){
                //implementação do método liberarNotas()
        }

        public float obterSaldoCaixa(){
                //implementação do método obterSaldoCaixa()
        }

        public boolean validarSenha(int pwd){
                //implementação do método validarSenha()
        }
}
```

O estado de um caixa pode ser definido através de dois atributos: o saldo do caixa, e a senha do seu administrador. A seguir é apresentada a definição desses atributos:

```
private float saldoCaixa;     // saldo no caixa, em R$
private int senhaCaixa;
```

O método recarregar(int pwd)

Implemente esse método, que verifica se a senha informada está correta e em caso afirmativo, define o saldo do caixa como R$1.000,00.

O método obterSaldoCaixa()

Implemente esse método, que retorna o saldo do caixa

O método validarSenha(int pwd)

Implemente esse método, que deve retornar true, se a senha estiver correta, e false caso contrário.

O método liberarNotas(int qtd)

Implemente esse método que, além de liberar notas, é responsável por reduzir o valor total das notas emitidas do saldo do caixa. Cada nota liberada deve ser representada pela impressão da string “===/ R$10,00 /===>”.

3.4 Especificação da Classe ContaCor

```
public class ContaCor {
        //Constantes
           //definição de constantes

        //Atributos
          // definição de atributos

    //Operacoes
    public ContaCor(String titular, float saldoAtual, int
numConta, int senha) {
       //implementalçao do construtor
    }
         public float obterSaldo(int pwd){
             //Implementação do método obterSaldo()
         }
         public boolean debitarValor(String hist, float val,
int pwd){
                  //implementação do método debitarValor()
         }
}
```

Foram definidas duas constantes para representar os possíveis modos de operação de uma conta corrente: ATIVA, equivalente ao valor inteiro 1 e ENCERRADA, equivalente ao valor inteiro 2.

```
public static int ATIVA = 1;
public static int ENCERRADA = 2;
```

O estado de uma conta corrente pode ser definido através de nove atributos, que definem os dados de uma conta:

```
private int estado;     // 1=Ativa, 2=Encerrada
private String titular; // nome do titular
private int numConta;   // número da conta
private int senha;      // senha
private float saldoAnterior; // saldo anterior
private String historico[];  // históricos e
private float valorLanc[];   // valores dos últimos
                              // lançamentos > 0 p/ créditos;
                             //< 0 p/ débitos
private int ultLanc;         // topo dos vetores acima
private float saldoAtual;    // saldo atual da conta
```

O método obterSaldo(int pwd)

Implemente esse método que verifica a validade da senha e o estado da conta. Se a senha estiver correta e a conta estiver ATIVA, retorna o saldo da conta, caso contrário, retorna -1.

O método debitarValor(String hist, float val, int pwd)

Implemente esse método que, após a verificação da senha e a verificação do estado da conta, que deve estar ATIVA, debita o valor informado do saldo da conta. O método retorna true, se a operação foi realizada com sucesso e false, caso contrário. Vale a pena lembrar que a conta deve manter o histórico das últimas transações. Sendo assim, atualize o atributo historico com o valor do parâmetro hist.

O construtor da classe

O método ContaCor(...) implementa o construtor da classe ContaCor, realizando a inicialização dos atributos da classe:

```
this.estado = ContaCor.ATIVA; // A conta se torna ativa,
                              //podendo receber lançamentos.
this.titular = titular;
this.saldoAtual = saldoAtual;
this.numConta = numConta;
this.senha = senha;
this.ultLanc=0;        // A conta sem nenhum lançamento.
this.historico=new String[11];  // cria vetores ...
this.valorLanc=new float[11];   // ... com 11 elementos
```

Compilando a classe ContaCor

Após criado o arquivo ContaCor.java contendo toda a definição da classe podemos compilá-la usando o comando:

```
javac ContaCor.java
```

3.5 Definição da Classe CadastroContas

- Nesse exemplo iremos simular a existência de um banco de dados que será implementado pela classe CadastroContas.
- Para este exemplo, essa classe define apenas um método, buscaConta, responsável por procurar um objeto do tipo ContaCor.
- Com o fim de facilitar a realização de testes, três contas são criadas pelo construtor da classe CadastroContas quando está é instanciada.

```
class CadastroContas {
  private ContaCor c[];  // vetor de contas
  CadastroContas () {  // método construtor
    c=new ContaCor[4];
    c[1]=new ContaCor("Ursula",500,1,1);
    System.out.println
      ("Criada conta 1 senha 1 com 500,00");
    c[2]=new ContaCor("Mia",500,2,2);
    System.out.println
      ("Criada conta 2 senha 2 com 500,00");
    c[3]=new ContaCor("Alfredo",500,3,3);
    System.out.println
      ("Criada conta 3 senha 3 com 500,00");
  }
  ContaCor buscaConta (int numcta) {
    if (numcta<1 || numcta>3)  // apenas 3 contas no BD
      return(null);
    else
      return(c[numcta]);
  }
}
```

Compilando a classe CadastroContas

Uma vez que tenha sido criado o arquivo CadastroContas.java contendo a definição da classe, podemos compilá-la usando o comando:

```
javac CadastroContas.java
```

Não deverá haver erro na compilação. Caso haja alguma mensagem de erro confira o conteúdo do arquivo com o código apresentado ao longo do texto e compile-o novamente após efetuar as correções necessárias.

Devemos agora compilar novamente as classes ControladorCaixa e TrmCxAut, para verificar se os erros gerados anteriormente são todos eliminados com a definição da classe CadastroContas. Como a classe TrmCxAut contém referência para a classe ControladorCaixa, basta compilarmos TrmCxAut, pois a classe controladora será compilada automaticamente.

4- Teste da aplicação

Para testar a aplicação desenvolvida vamos utilizar um pequeno programa que cria o banco de dados e inicia a operação do terminal de caixa:
Crie um arquivo de nome Principal.java contendo:

```
class Principal {
    public static void main (String[] args) {
       //Instanciacao do caixa automatico
        TrmCxAut meuCaixaAut = new
TrmCxAut(123,TrmCxAut.MODO_SUPERVISOR);

       //utilizacao do caixa
       meuCaixaAut.iniciarOperacao();
    }
}
```

Em seguida compile o programa usando o comando:

```
javac Principal.java
```

Não havendo erros, a aplicação pode ser executada com:

```
java Principal
```

Cenário de execução:

1. Recarga do caixa;
2. Alternância do modo de operação;
3. Consulta de saldo da conta 1;
4. Saque de R$ 50,00 da conta 1;
5. Consulta de saldo da conta 1;
6. Sair do sistema.

```
Criada conta 1 senha 1 com 500,00
Criada conta 2 senha 2 com 500,00
Criada conta 3 senha 3 com 500,00
Entre com opcao: 3 = recarrega, 8=modo cliente, 9=sai
3
Entre com senha
123
Entre com opcao: 3 = recarrega, 8=modo cliente, 9=sai
8
Entre com senha
123
Entre com opcao: 1 = consulta saldo, 2 = saque, 8=modo supervisor, 9=sai
1
```

```
Entre com numero da conta
1
Entre com senha
1
Saldo atual: 500.0
Entre com opcao: 1 = consulta saldo, 2 = saque, 8=modo supervisor, 9=sai
2
Entre com número da conta
1
Entre com senha
1
Entre com valor
50
===/ R$10,00 /===>
===/ R$10,00 /===>
===/ R$10,00 /===>
===/ R$10,00 /===>
===/ R$10,00 /===>
Pode retirar o dinheiro
Entre com opcao: 1 = consulta saldo, 2 = saque, 8=modo supervisor, 9=sai
1
Entre com numero da conta
1
Entre com senha
1
Saldo atual: 450.0
Entre com opcao: 1 = consulta saldo, 2 = saque, 8=modo supervisor, 9=sai
9
```