Orientação a Objetos e Java

Sérgio Soares
sergio@dsc.upe.br

Exceções

Objetivo

Depois desta aula você será capaz de desenvolver sistemas robustos, notificando e tratando casos de erro na execução de métodos através do mecanismo de exceções de Java.

Exceções

Leitura prévia essencial

- Capítulo 12 do livro *Java: how to program* (de Harvey e Paul Deitel)

Classe de Contas: Definição

```
public class Conta {
  private String numero;
  private double saldo;
  ...
  public void debitar(double valor) {
    saldo = saldo - valor;
  }
}
```

Como evitar débitos acima do limite permitido?

Desconsiderar Operação

```
public class Conta {
  private String numero;
  private double saldo;
  ...
  public void debitar(double valor) {
    if (valor <= saldo)
      saldo = saldo - valor;
  }
}
```

Desconsiderar Operação

- Problemas:
  - quem solicita a operação não tem como saber se ela foi realizada ou não
  - nenhuma informação é dada ao usuário do sistema

Mostrar Mensagem de Erro

```
public class Conta {
  private String numero;
  private double saldo;
  ...
  public void debitar(double valor) {
    if (valor <= saldo)
      saldo = saldo - valor;
    else
      System.out.print("Saldo Insuficiente");
  }
}
```

Mostrar Mensagem de Erro

- Problemas:
  - informação é dada ao usuário do sistema, mas nenhuma sinalização de erro é fornecida para métodos que invocam **`debitar`**
  - supõe que a interface com o usuário é modo texto
  - cria uma forte dependência entre a classe **`Conta`** e sua interface com o usuário
  - entrelaçamento entre do código da camada de negócio com o código da camada de interface com o usuário

Retornar Código de Erro

```
public class Conta {
  private String numero;
  private double saldo;
  ...
  public boolean debitar(double valor) {
    boolean r  = false;
    if (valor <= saldo) {
      saldo = saldo - valor;
      r  = true;
    } else r = false;
    return r;
  }
}
```

Retornar Código de Erro

- Problemas:
  - dificulta a definição e o uso do método
  - não torna a causa ou o tipo do erro explícito
  - métodos que invocam **`debitar`** têm que testar o resultado retornado para decidir o que deve ser feito
  - A dificuldade é maior para métodos que retornam valores:
    - e se **`debitar`** já retornasse um outro valor qualquer? O que teria que ser feito?

Código de Erro: Problemas

```
public class CadastroContas {
  ...
  public int debitar(String numero,
                     double valor) {
    int erro = 0;
    Conta c = contas.procurar(numero);
    if (c != null) {
      boolean b = c.debitar(valor);
      if (b) erro = 0;
      else erro = 2;
    } else erro = 1;
    return erro;
  }
}
```

Exceções

- Ao invés de códigos, teremos exceções...
- São objetos comuns, portanto têm que ter uma classe associada
- Classes representando exceções herdam e são subclasses de **`Exception`** (pré-definida)
- Define-se subclasses de **`Exception`** para
  - oferecer informações extras sobre a falha, ou
  - distinguir os vários tipos de falhas

Definindo Exceções

```
public class SIException extends Exception {
  private double saldo;
  private String numero;
  public SIException(double saldo,
                     String numero) {
    super ("Saldo Insuficiente!");
    this.saldo = saldo;
    this.numero = numero;
  }
  public double getSaldo() {
    return saldo;
  }
  ...
}
```

Definindo Métodos com Exceções

```
public class Conta {
  ...
  public void debitar(double valor)
        throws SIException {
    if (valor <= saldo)
      saldo = saldo - valor;
    else {
      SIException e;
      e = new SIException(saldo,numero);
      throw e;
    }
  }
}
```

Definindo e Levantando Exceções

- ***Res metodo*(*Pars*) throws *E1,...,EN***
- Todos os tipos de exceções levantadas no corpo de um método devem ser declaradas na sua assinatura
- Levantando exceções: **throw** *obj-exceção*
- Fluxo de controle e exceção são passados para a chamada do código que contém o comando **throw**, e assim por diante...

Usando Métodos com Exceções

```
public class Conta {
  ...
  public void transferir(Conta conta,
                         double valor)
       throws SIException {
    this.debitar(valor);
    conta.creditar(valor);
  }
}
```

Exceções levantadas indiretamente também devem ser declaradas!

Usando e Definindo Métodos com Exceções

```
public class CadastroContas {
  ...
  public void debitar(String numero,
                      double valor)
       throws SIException, CNEException {
    Conta c = contas.procurar(numero);
    c.debitar(valor);
  }
}
```

Tratando Exceções

```
Antes...
try {
  ...
  banco.debitar("123-4",90.00);
  ...
} catch (SIException e) {
  System.out.print(e.getMessage());
  System.out.print(" Conta/saldo: ");
  System.out.print(e.getNumero()+
                   " / " + e.getSaldo());
} catch (CNEException e) {...}
Depois...
```

Tratando Exceções

- A execução do **`try`** termina assim que uma exceção é levantada
- O *primeiro* **`catch`** de um supertipo da exceção é executado e o fluxo de controle passa para o código seguinte ao último **`catch`**
- Se não houver nenhum **`catch`** compatível, a exceção e o fluxo de controle são passados para a chamada do código (método) com o **`try/catch`**

Tratando Exceções: Forma Geral

```
try {
  ...
} catch (E1 e1) {
  ...
}
...
} catch (En en) {
  ...
} finally {
  ...
}
```

O bloco finally é opcional desde que exista pelo menos um bloco catch

Tratando Exceções

- O bloco **`finally`** é sempre executado mesmo
  - após a terminação normal do **`try`**
  - após a execução de um **`catch`**
  - quando não existe nenhum **`catch`** compatível
- Quando o **`try`** termina normalmente ou um **`catch`** é executado, o fluxo de controle é passado para o código seguindo o bloco **`finally`** (depois deste ser executado)

Exceções no Cadastro de Contas

```
public class CadastroContas {
  ...
  public void cadastrar(Conta conta)
        throws CJCException,
               IllegalArgumentException {
    if (conta != null) {
      String numero = conta.getNumero();
      if (!contas.existe(numero))
        contas.inserir(conta);
      else
        throw new CJCException(numero);
    } else
      throw new IllegalArgumentException();
    }
```

RuntimeException não precisam ser declaradas/tratadas

Exceções no Cadastro de Contas

```
  public void debitar(String numero,
                      double valor)
        throws SIException, CNEException {
    Conta c = contas.procurar(numero);
    c.debitar(valor);
  }
}
```

Exceções no Conjunto de Contas

```
public class RepositorioContasArray
          implements RepositorioContas {
  ...
  public static final
      IllegalArgumentException e =
      new IllegalArgumentException();
  public void inserir(Conta conta)
        throws IllegalArgumentException {
    if (conta != null) {
      contas[indice] = conta;
      indice = indice + 1;
    } else throw e;
  }
```

Exceções no Conjunto de Contas

```
public Conta procurar(String numero)
      throws CNEException {
  Conta conta = null;
  int i = this.procurarIndice(numero);
  if (i == indice)
    throw new CNEException(numero);
  else
    conta = contas[i];
  return conta;
 }
}
```

Exercícios

- Altere o método saldo da classe **`CadastroContas`** para gerar uma exceção quando alguém quiser saber o saldo de uma conta não cadastrada.
- Com base na nova definição de **`Conta`** usando exceções, indique o que precisa ser alterado nas classes **`Poupanca`** e **`ContaEspecial`** para que elas também usem exceções.

Exceções

Resumo

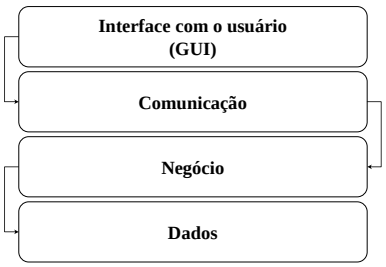
- Alternativas para notificar e tratar casos de erro em Java
- Exceções: objetos e classes
- Comando `throw`, cláusula `throws` e exceções verificadas
- Comando `try-catch-finally`

Exceções

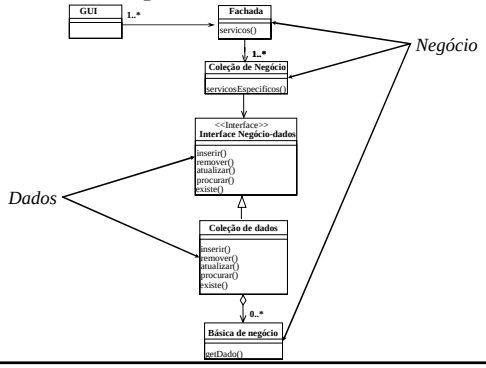
Leitura adicional

- Capítulo 12 do livro *Java: how to program* (de Harvey e Paul Deitel)
- Capítulo 9 do livro *Thinking in Java* (de Bruce Eckel)
- Seções 5.5 a 5.8 do livro *A Programmer's Guide to Java Certification* (de Khalid Mughal e Rolf Rasmussen)

Vendo o código como um bolo... com várias camadas!



Arquitetura de software



Classes Básicas de Negócio

```
public class Conta {
  private double saldo;
  private String numero;
  private Cliente correntista;
  ...
  public void creditar(double valor) {
    saldo = saldo + valor;
  }
}
```

Cliente, Livro, Animal, Veiculo

Interfaces Negócio-Dados

```
public interface RepositorioContas {
  public void inserir(Conta conta)
      throws RepositorioException,
             ContaInvalidaException;
  public void atualizar(Conta conta)
      throws RepositorioException,
             ContaInvalidaException,
             ContaNaoEncontradaException;
  public void remover(String numero);
      throws RepositorioException,
             ContaInvalidaException,
             ContaNaoEncontradaException;
  ...
}
```

Classes Coleção de Dados

```
public class RepositorioContasArray
      implements RepositorioContas {
  ...
  public void inserir(Conta conta)
      throws ContaInvalidaException {
    if (conta == null)
      throw new ContaInvalidaException();
    contas[indice] = conta;
    indice = indice + 1;
  } ...
}
```

Classes Coleção de Dados Persistentes

```
public class RepositorioContasBDR
      implements RepositorioContas {
  ...
  public void inserir(Conta conta)
      throws ContaInvalidaException,
             RepositorioException {
    try {
      ...
    } catch(SQLException ex) {
      throw new RepositorioException(ex);
    }
  } ...
}
```

Classes Coleção de Negócio

```
public class CadastroContas {
  private RepositorioContas contas;
  ...
  public void cadastrar(Conta conta)
      throws ContaInvalidaException,
             RepositorioException,
             ContaJaCadastradaException {
    if (!contas.existe(conta.getNumero())) {
      contas.inserir(conta);
    }
    else throw new ContaJaCadastradaException();
  } ...
}
```

Classe Fachada

```
public class Banco {
  private CadastroClientes clientes;
  public void cadastrar(Conta conta)
      throws ContaInvalidaException,
             RepositorioException,
             ContaJaCadastradaException
             CorrentistaNaoCadastradoException {
    Cliente c = conta.getCorrentista();
    if (clientes.existe(c.getCodigo())
      contas.cadastrar(conta);
    else
      throw new CorrentistaNaoCadastradoException();
  } ...
}
```