Tratamento de exceções em Java

Prof. Gustavo Wagner

(Alterações)

Prof. Tiago Massoni

(Slides Originais)

Desenvolvimento de Sistemas

FATEC-PB

© Centro de Informática, UFPE

Conta: declaração

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

Como evitar débitos acima do limite permitido?

1. Desconsiderar Operação

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

2. 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 negativo");
  }
}
```

- Mistura indesejada
- Como avisar quem chamou?

3. 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;
    }
    return r;
  }
}
```

3. Retornar Código de Erro

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

2 tipos de erros possíveis - tende a piorar...

3. Retornar Código de Erro

- Não retorna 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 já retornam valores
  - e se **`debitar`** já retornasse um outro valor qualquer? O que teria que ser feito?

Exceções

- Ao invés de códigos, teremos exceções...
- São objetos comuns, a partir de uma classe associada
- Classes representando exceções são subclasses de **`Exception`** (API de Java)
- Definiremos subclasses de **`Exception`**
  - oferecer informações extras sobre a falha, ou
  - distinguir os vários tipos de falhas

Solução: exceções em Java

Todas as exceções estendem java.lang.Exception

```
public class SaldoNegativoException extends

    Exception {
  public SaldoNegativoException(String num) {
    super ("Saldo Insuficiente em:"+num);
  }
}
```

Define o construtor com mensagem

Métodos e exceções

Anuncia "eu posso lançar essa daqui!!"

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

Ops, aconteceu!
Criar objeto e lançar

Métodos e exceções

```
public class Conta {
   ...
 public void transferirDe(Conta c, double v)
                    throws SaldoNegativoException {
    this.debitar(v);
    c.creditar(v);
 }
}
```

Exceções lançadas indiretamente também devem ser avisadas

Tratamento de exceções

```
FachadaBanco banco = new FachadaBanco(...);
try {
   ...
   banco.debitar("123-4",90.00);
   ...
} catch (SaldoNegativoException e) {
      System.out.println(e.getMessage());
}
{...outros blocos catch...}
```

Se nos interessa, vamos "prender" a exceção!

Tratando exceções

- A execução do `try` termina assim que uma exceção é lançada
- O primeiro `catch` de uma exceção é executado e o fluxo de controle passa para o código seguinte ao último `catch`
- Se não houver nenhum `catch` compatível, a exceção passa para quem chamou este método com `try/catch`

Tratando exceções: forma geral

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

O bloco finally é sempre executado, qualquer que seja o resultado

Tratando exceções

- O bloco **`finally`** é sempre executado
  - após a terminação normal do **`try`**
  - após a execução de um **`catch`**
  - quando não existe nenhum **`catch`** compatível
- Quando o **`try`** termina sem exceções ou um **`catch`** é executado
  - fluxo de controle é passado para o bloco **`finally`**
  - Depois continua normalmente para o próximo comando

Exceções nas camadas: básicas

```
public class FachadaBanco {...
  public void debitar(String num, double valor)
              throws SaldoNegativoException{
      Conta c = repContas.procurar(num);
      c.debitar(valor);
  }
}
```

Deu o aviso!

```
public class InterfaceTexto {...
FachadaBanco banco = new FachadaBanco(...);
try {
  ...
  banco.debitar("123-4",90.00);
  ...
} catch (SaldoNegativoException e) {
      System.out.println(e.getMessage());
}
```

Exceções: repositórios

Antes

```
public class RepositorioContasArray { ...
public Conta procurar(String numero) {
 int ind = procurarIndice(numero);
 if (ind != -1)
    return contas[ind];
 else return null;
}
```

Depois

```
public class RepositorioContasArray { ...
public Conta procurar(String numero)
                    throws ObjetoNaoEncontradoException {
 int ind = procurarIndice(numero);
 if (ind != -1)
   return contas[ind];
 else
   throw new ObjetoNaoEncontradoException("Conta
inexiste");
}
```

Exceções: repositórios

Antes

```
public class FachadaBanco {...
  public void debitar(String num, double valor)
              throws SaldoNegativoException{
      Conta c = repContas.procurar(num);
      c.debitar(valor);
  }
```

Depois

```
public class FachadaBanco {...
  public void debitar(String num, double valor)
              throws SaldoNegativoException,
                     ObjetoNaoEncontradoException{
      Conta c = repContas.procurar(num);
      c.debitar(valor);
  }
```

Exceções: fachada

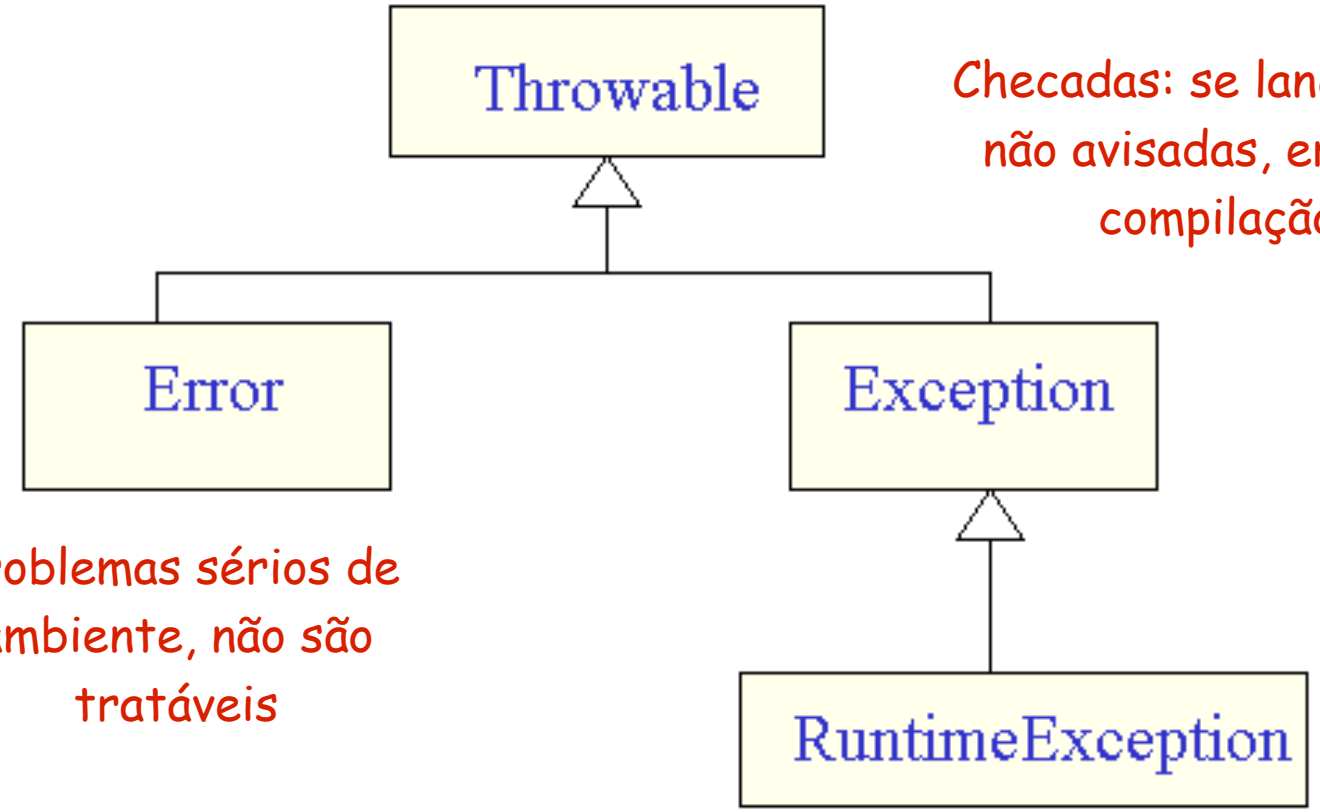
Antes

```
public class FachadaBanco {...
 public void cadastrarCliente (Cliente cli){
   boolean achou = repClientes.existe(cli.getCpf());
   if (!achou)
      repClientes.inserir(cli);
 }
```

Depois

```
public class FachadaBanco {...
 public void cadastrarCliente (Cliente cli)
                    throws ObjetoJaCadastradoException {
  boolean achou = repClientes.existe(cli.getCpf());
  if (!achou)
   repClientes.inserir(cli);
  else
   throw new ObjetoJaCadastradoException("Cliente existe");
 }
```

Exceções em Java: hierarquia



Checadas: se lançadas e não avisadas, erro de compilação

Problemas sérios de ambiente, não são tratáveis

Não-checadas: não tem problema de compilação; erros que não deveriam ir para o usuário

Aula Prática
Exceções

Objetivos da prática

- Modificar sistema bancário em camadas para incluir tratamento de exceções
- Exceções são lançadas onde ocorrem, mas tratamento é feito apenas na GUI

Passo 1

- Colocar exceções no pacote das classes que elas lançam;
- Criar as exceções apropriadas, de acordo com as necessidades do projeto;

Passo 2

- SaldoNegativoException
- Lançar a exceção no método debitar de Conta
  - Não esqueça do anúncio
- Atualizar classes que chamam este método
- Na InterfaceTexto, tratar a exceção ao chamar debitar
  - Macete do Eclipse para o try,catch

Passo 3

- ObjetoNaoEncontradoException
- ObjetoJaCadastradoException
- São lançadas nos Repositórios ou Fachada
  - Repositorio: procurar, atualizar
  - Fachada: cadastrarXXX
  - Atualizar os que chamam
- Exceções tratadas apenas na camada GUI
  - Único lugar onde podemos mostrar uma mensagem ao usuário