Programação Orientada à Objetos (POO)

CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. João Paulo Aramuni

UNIVERSIDADE FUMEC

Sumário

* **Exceções**
  * Declaração de exceções
  * Manipulação de exceções
  * Tratamento de exceções

Exceções

* **Exceções**
  * Considere o seguinte trecho de código:

```
public class Estoque {
private int codProduto;
private int qtdeEstoque;
    public void decrementaEstoque (int qtde) {
                qtdeEstoque = qtdeEstoque - qtde;
    }
}
```

Exceções

* **Exceções**
  * O que fazer se o estoque não for suficiente?
  * Vamos avaliar três possibilidades:
    * Desconsiderar a operação;
    * Mostrar mensagem de erro;
    * Retornar código de erro.

Exceções

* **Exceções**
  * Opção 1: Desconsiderar operação.

```
public class Estoque {
private int codProduto;
private int qtdeEstoque;
public void decrementaEstoque (int qtde) {
    if (qtdeEstoque >= qtde)
                qtdeEstoque = qtdeEstoque - qtde;
    }
}
```

Exceções

* **Exceções**
  * Problema: Não há como saber se a operação foi realizada.
  * Nenhuma informação é retornada ao **cliente**.

Exceções

* **Exceções**
  * Opção 2: Mostrar mensagem de erro.

```
public class Estoque {
private int codProduto;
private int qtdeEstoque;
public void decrementaEstoque (int qtde) {
    if (qtdeEstoque >= qtde)
                qtdeEstoque = qtdeEstoque - qtde;
                else System.out.println("Estoque Insuficiente!");
    }
}
```

Exceções

* **Exceções**
  * Gera dependência entre a classe e sua interface de usuário.
  * Não retorna informação ao **cliente.**

Exceções

- **Exceções**
  - Opção 3: Retornar código de erro

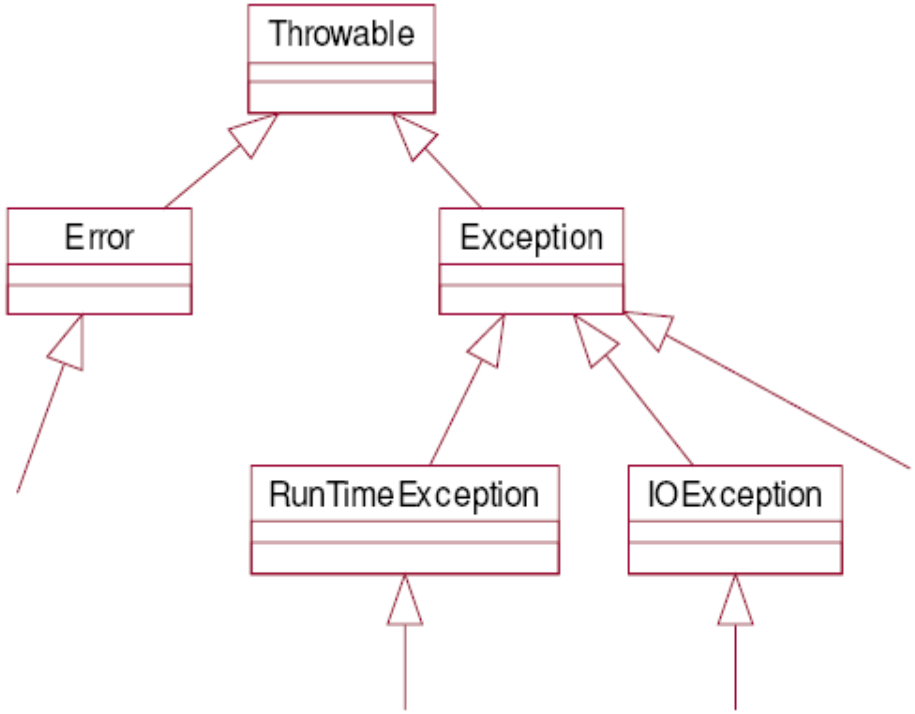
```
public class Estoque {
private int codProduto;
private int qtdeEstoque;
public boolean decrementaEstoque (int qtde) {
    if (qtdeEstoque >= qtde){
                qtdeEstoque = qtdeEstoque - qtde;
                return true; }
                else return false;
    }
}
```

Exceções

* **Exceções**
  * Complica definição e uso do método (cliente têm que fazer testes);
  * Pior para métodos que já retornam valores.

Exceções

* **Exceções**
  * Solução:
    * Usar esquema de tratamento de exceções.
    * Hierarquia de exceções em Java:



Exceções

* **Exceções**
  * **Error**
    - Descreve erros internos. Ex: falta de memória, disco cheio.
    - Não se deve lançar um objeto desse tipo (Error).
    - Pouco se pode fazer se um erro interno ocorre, além de notificar o usuário e tentar finalizar o programa.
  * **Exception**
    - RunTimeException
      - Ocorre porque houve um erro de programação (culpa do programador).
      - Ex: conversão explícita de tipo (cast), acesso a elemento de array além dos limites, acesso de apontador nulo.
    * Outras exceções
      - Ex: tentar ler além do final de um arquivo, tentar abrir URL incorreta.

Exceções

* **Exceções**
  * As exceções que derivam da classe **Error** ou da classe **RunTimeException** são chamadas exceções **não verificadas**. Todas as demais são chamadas exceções verificadas.
  * Um método precisa declarar todas as exceções verificadas que ele lança. Palavra-chave throws no cabeçalho.

```
public void gravaDados()  throws  IOException {
ObjectOutputStream out = new ObjectOutputStream(
 new FileOutputStream(”rh.dat"));
out.writeObject(funcionarios);
out.close();
}
```

  * Se o método não conseguir criar o arquivo ou gravar os dados, ocorrerá uma exceção do tipo **IOException**, que deverá ser tratada.

Exceções

* **Exceções**
  * A hierarquia de exceções possui várias classes para diversos tipos de exceções que podem ser lançadas pelos programas.
  * Para lançar uma exceção, use a palavra-chave throw.

```
public String lerDados (BufferedReader ent)  throws EOFException {
. . .
while ( . . . ) {
 if (ch == -1)
                if (n < compr)
                  throw  new  EOFException();
} }
```

  * A classe **EOFException** sinaliza que ocorreu um fim de arquivo inesperado durante a entrada de dados.

Exceções

* **Exceções**
  * Como criar suas próprias classes de Exceção?
    * Um programa pode ter um problema que não está descrito adequadamente em nenhuma das classes de exceção padrão.
    * Para criar um classe de exceção basta derivá-la de Exception ou de uma classe descendente de Exception.
    * É habitual criar um construtor padrão e um construtor que contenha uma mensagem detalhada.
    * Construtores/método **getMessage** da classe Throwable:
      * Throwable() : constrói um novo objeto sem mensagem detalhada.
      * Throwable (String mensagem) : constrói um novo objeto com a mensagem detalhada especificada.
      * String getMessage( ) : este método obtém a mensagem detalhada do objeto Throwable.

Exceções

* **Exceções**
  * Lançamento de uma exceção criada:

```
public class Estoque {
        private int codProduto;
        private int qtdeEstoque;
         ...
        public void decrementaEstoque (int qtde) throws
        EIException {
        if (qtdeEstoque >= qtde)
                qtdeEstoque = qtdeEstoque - qtde;
        else  throw new  EIException (codProduto,qtdeEstoque);
        }
}
```

Exceções

* **Exceções**
  * Criação de uma classe de exceção: **EIException**

```
public class EIException  extends  Exception {
        private int codProd;
        private int qtdeEst;
        public EIException ( ) {
        super (“Estoque Insuficiente!”); }
        public EIException (int cod, int qtde) {
        super (“Estoque Insuficiente!”);
        codProd = cod;
        qtdeEst = qtde; }
        public int getCodProduto( ) {
                return codProd; }
        public int getQtdeEstoque( ) {
                return qtdeEst; }
}
```

Exceções

* **Exceções**
  * Exceções levantadas indiretamente também devem ser tratadas:

```
public class Pedido {
. . .
public void adicionaItem (Estoque est, int qtde) throws
 EIException {
. . .
 est.decrementaEstoque (qtde); // <-lança a exceção
. . .
}
}
```

Exceções

* **Exceções**
  * Se não se desejar criar uma classe de exceção própria pode-se levantar uma exceção com uma mensagem personalizada através de uma classe pré-existente.

```
public class Estoque {
 private int codProduto;
 private int qtdeEstoque;
 public void decrementaEstoque (int qtde) throws Exception {
 if (qtdeEstoque >= qtde)
 qtdeEstoque = qtdeEstoque - qtde;
 else  throw  new  Exception (“Estoque  Insuficiente!”);
}}
```

  * Mas lembre-se que a classe Exception é muito genérica e captura qualquer exceção !

Exceções

* **Exceções**
  * Criar uma classe de exceção própria declarar (com throws) e levantar (com throw) a exceção:

```
public class Estoque {
    private int codProduto;
    private int qtdeEstoque;
    public void decrementaEstoque (int qtde) throws  EIException {
    if (qtdeEstoque >= qtde)
    qtdeEstoque = qtdeEstoque - qtde;
    else throw new  EIException (codProduto, qtdeEstoque);
    }
}
```

Exceções

* **Exceções**
  * Captura de exceções:
    * As exceções devem ser capturadas e tratadas (verificadas) adequadamente.
    * Se ocorrer uma exceção e esta não for capturada em nenhum lugar, o programa termina mostrando uma mensagem de erro.
    * Para capturar uma exceção, especifica-se um bloco try, catch (p/ verificar) e finally (bloco de finalizações opcional).

```
try {  // Capturar a exceção
        comando1;
        comando2;   . . . comandon;
        } catch ( TipoExceção1  e1 ) { tratamento de  e1 } //verificar
        catch ( TipoExceção2  e2 ) { tratamento de  e2 } . . .
        catch ( TipoExceçãon  en )  { tratamento de  en } //verificar
        finally { finalizações }  //bloco de finalizações
```

Exceções

* **Exceções**
  * Exemplo: Captura e Verificação de exceção

```
public class Interface {
. . .
public void insereItemPedido {
. . .
try {
    pedido.adicionaItem (est, qtde);
} catch ( EIException  eierro ) {
    System.out.println ( eierro.getMessage() +
                "  Produto: " + eierro.getCodProduto() +
                "  Estoque: " + eierro.getQtdeEstoque() );
   {
. . .
  {
  {
```

Exceções

* **Exceções**
  * Se o código dentro do bloco try não lançar nenhuma exceção
    * o programa executa o código dentro de try, **pula a cláusula catch**, executa o código dentro da cláusula finally se estiver presente e o restante do código.

  * Se qualquer parte do código dentro do bloco try lançar uma exceção da classe especificada na cláusula catch
    * o programa pula o restante do código do bloco try a partir do ponto onde a exceção foi lançada, executa o código da cláusula catch correspondente e, então, o código da cláusula finally.
      * Se a cláusula catch não lançar nenhuma exceção, o programa executa a primeira linha depois do bloco try. Senão, a exceção é lançada de volta para o chamador do método.

Exceções

* **Exceções**
  * Se código dentro do bloco **try** lançar uma exceção que não é capturada por nenhuma cláusula **catch**

    * o programa pula o restante do código do bloco try a partir do ponto onde a exceção foi lançada, executa o código na cláusula finally e lança a exceção de volta para o chamador do método.

Exceções

* **Exceções**
  * Para capturar múltiplas exceções
    * Usam-se cláusulas catch separadas, uma para cada tipo de exceção.
      * Deve-se colocar as classes mais específicas primeiro.

  * Exceções podem ser relançadas dentro da cláusula **catch**.

```
. . .
Graphics g = image.getGraphics();
try {
 código que pode lançar exceções
} catch (MalformedURLException e) {
 g.dispose();
 throw  e;
}
```

Exceções

- **Exceções**
  - Para capturar múltiplas exceções
    - Usam-se cláusulas catch separadas, uma para cada tipo de exceção.
      - Deve-se colocar as classes mais específicas primeiro.

```
try {
    icli.leDados();
 }
catch (ClassNotFoundException e) {
System.out.println("O arquivo de dados não é compatível. "+
 "Um novo arquivo será criado!");}
  catch (IOException e) {
System.out.println("Arquivo de dados não encontrado. "+
  "Um novo arquivo será criado!");
}
```

* **Exceções**

```
private void leDados()  throws  IOException, ClassNotFoundException {
    ObjectInputStream in = new ObjectInputStream(
    new FileInputStream("cliente.dat"));
    clientes = (Vector) in.readObject( );
    in.close();
    if (clientes.size()==0) {
          return;  // Lista está vazia?
     }
    Cliente cli = (Cliente) clientes.get(clientes.size()-1);
    Cliente.setCodCli (cli.getCodigo( )); // inicia atrib. estático
 }
```

Obrigado.

joaopauloaramuni@gmail.com
joaopauloaramuni@fumec.br

UNIVERSIDADE FUMEC
50 Anos