Tratamento de Exceções

SCC0604 - Programação Orientada a Objetos

Prof. Fernando V. Paulovich
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

8 de novembro de 2010

Introdução

- O código gerado até agora leva somente em consideração que tudo funcionará sempre corretamente, porém isso nem sempre é verdade - **erros irão acontecer!**

Introdução

- O código gerado até agora leva somente em consideração que tudo funcionará sempre corretamente, porém isso nem sempre é verdade - **erros irão acontecer!**

- Existem muitos meios de se tratar erros. Mais comumente, o código de tratamento de erro está misturado no meio do código de um sistema - os **erros são tratados** nos lugares do código onde os **erros podem acontecer**

Introdução

- O código gerado até agora leva somente em consideração que tudo funcionará sempre corretamente, porém isso nem sempre é verdade - **erros irão acontecer!**
- Existem muitos meios de se tratar erros. Mais comumente, o código de tratamento de erro está misturado no meio do código de um sistema - os **erros são tratados** nos lugares do código onde os **erros podem acontecer**
- O problema com essa abordagem é que o **código** se torna "**poluído**" com o processamento de erros

Introdução

- Em situações excepcionais, Java oferece uma forma de captura de erros chamada de **tratamento de exceções**

Introdução

- Em situações excepcionais, Java oferece uma forma de captura de erros chamada de **tratamento de exceções**
- A ideia por trás do tratamento de exceções é possibilitar aos programas **capturar e tratar erros**, em vez de deixá-los acontecer e simplesmente sofrer as consequências

Introdução

- O recurso de tratamento de exceções possibilita ao programador **remover o código de tratamento de erros da "linha principal" de execução** de um programa, melhorando a legibilidade e a possibilidade de modificar o programa

Introdução

- O recurso de tratamento de exceções possibilita ao programador **remover o código de tratamento de erros da "linha principal" de execução** de um programa, melhorando a legibilidade e a possibilidade de modificar o programa

- O tratamento de exceções é tipicamente usado em situações em que o **erro será tratado por uma parte diferente** do programa daquela **que descobriu o erro** - uma parte que possa tratar esse erro

Lançando Exceções

- Para **Informar o erro**, o método onde o mesmo ocorreu encapsula em um objeto a informação de erro e **lança esse objeto**; nesse momento, o método é encerrado

Lançando Exceções

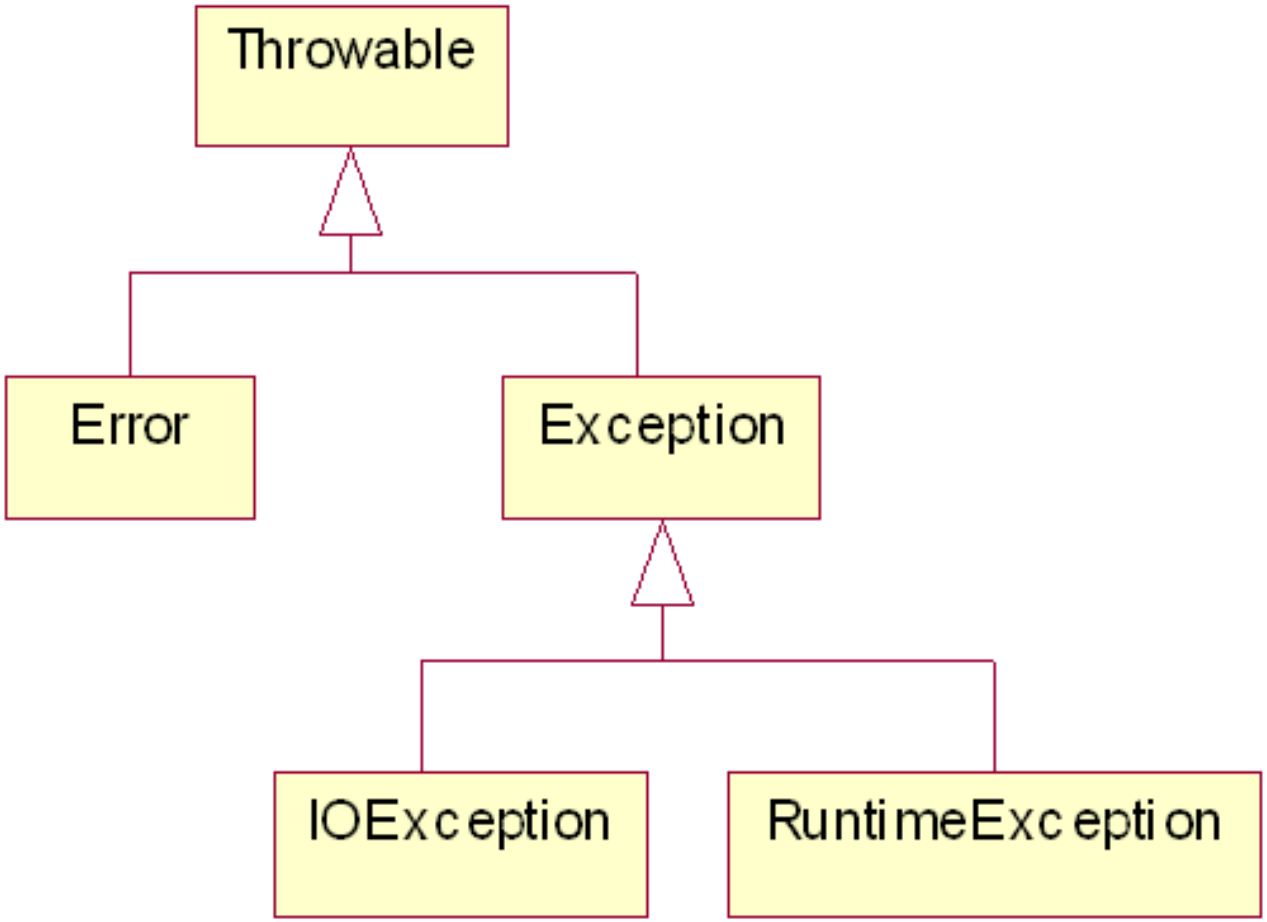
- Para **Informar o erro**, o método onde o mesmo ocorreu encapsula em um objeto a informação de erro e **lança esse objeto**; nesse momento, o método é encerrado

- Em Java, um objeto de exceção é sempre uma instância de uma classe derivada de **Throwable**

Hierarquia de Exceções

- Na verdade, Java define uma hierarquia de exceções, onde **Throwable** é a base, sendo derivada em duas outras classes **Error** e **Exception**
- Uma exceção **Error** é um problema interno do sistema - não deve ser lançada explicitamente
- A hierarquia **Exception**, por sua vez, se divide em dois ramos
  - **RuntimeException** para erros de programação
  - **IOException** para erros de entrada de dados

Hierarquia de Exceções



Como Anunciar que um Método pode Lançar uma Exceção

- As exceções que um método pode lançar são listadas junto ao cabeçalho desse método
- Nessa lista, exceções derivadas de **Error** e **RuntimeException** não devem ser anunciadas

```
1 public void setData(int dia, int mes, int ano) throws IOException {
2   ...
3 }
```

Como Anunciar que um Método pode Lançar uma Exceção

- Quando um método de uma classe declara que ele lança uma exceção que seja uma instância de uma classe, então ele pode lançar uma exceção daquela classe ou de qualquer de suas subclasses

Como Lançar uma Exceção

- Para se lançar uma exceção, a palavra-chave **throw** é usada

```
1  public class Data {
2    public void setData(int dia, int mes, int ano) throws IOException {
3      ....
4      throw new IOException();
5    ....
6    }
7  }
```

Como Lançar uma Exceção

- É possível no momento de lançar uma exceção **IOException**, passar uma mensagem de erro

```
1  public class Data {
2    public void setData(int dia, int mes, int ano) throws IOException {
3      ....
4      if(mes < 0 || mes > 12) {
5        throw new IOException("Mês inválido");
6      }
7    ....
8    }
9  }
```

Como criar uma Classe de Exceção

- Caso nenhuma classe de exceção existente tiver um significado ligado à exceção que se queira lançar, uma classe de exceção que a represente pode ser criada

- Para isso é só fazer essa classe derivar de **Exception** ou qualquer outra abaixo da hierarquia de **Exception**

Como criar uma Classe de Exceção

```
1  public class InvalidValueException extends IOException {
2    public InvalidValueException () {
3    }
4
5    public InvalidValueException (String msg) {
6      super(msg);
7    }
8  }
```

Usando a Classe Criada

```
1  public class Data {
2    public void setData(int dia, int mes, int ano) throws ↩
         InvalidValueException {
3      ....
4      if(mes < 0 || mes > 12) {
5        throw new InvalidValueException("Mês inválido");
6      }
7    ....
8    }
9  }
```

Capturando Exceções

- Após lançar uma exceção, será necessário tratá-la em algum lugar do programa, caso contrário o programa (não gráfico) irá terminar
- Para se capturar especifica-se um bloco **try/catch**

Capturando Exceções

- O bloco mais simples é o seguinte

```
1 try {
2   //código
3 } catch(TipoExcecao e) {
4   //tratamento para esse tipo de exceção
5 }
```

Capturando Exceções

- Se o código dentro do **try** lançar uma exceção do tipo especificado, então
  - O programa pula o restante do código no bloco **try**
  - O programa executa o código de manipulação dentro da cláusula **catch**

Capturando Exceções

- Se o código dentro do **try** lançar uma exceção do tipo especificado, então
  - O programa pula o restante do código no bloco **try**
  - O programa executa o código de manipulação dentro da cláusula **catch**

- Se o código dentro do **try** não lançar nenhuma exceção o bloco **catch** é pulado

Capturando Exceções

- Se o código dentro do **try** lançar uma exceção do tipo especificado, então
  - O programa pula o restante do código no bloco **try**
  - O programa executa o código de manipulação dentro da cláusula **catch**
- Se o código dentro do **try** não lançar nenhuma exceção o bloco **catch** é pulado
- Se a exceção lançada dentro do **try** não tiver especificada no **catch**, o método finaliza - outro bloco **try** deve tratar essa exceção, caso contrário, o programa termina

Exemplo

```
1 try {
2   Data d = new Data();
3   d.setData(1,13,2004);
4 } catch(InvalidValueException e) {
5   System.out.println(e.getMessage);
6 }
```

Capturando Exceções

- Caso seja usado um método que lance uma exceção dentro de outro, mas não se queira tratar tal exceção, esse novo método deve declarar que essa exceção pode ser lançada

```
1 public void teste() throws InvalidValueException {
2   ...
3   Data d = new Data();
4   d.setData(1,1,2004);
5   ...
6 }
```

Como Capturar Múltiplas Exceções

- Podemos capturar vários tipos de exceções diferentes dentro de um bloco **try**, para isso cláusulas **catch** para cada um desses tipos devem ser usadas

```
1 try {
2   ...
3 } catch(ExcecaoTipo1 e1) {
4   ...
5 } catch(ExcecaoTipo2 e2) {
6   ...
7 }
```

A cláusula *finally*

- Uma cláusula **finally** sempre é executada em um método, sendo lançada ou não a exceção

```
1  Graphics g = image.getGraphics();
2
3  try {
4    ...
5  } catch(IOException e) {
6    ...
7  } finally {
8    g.dispose();
9  }
```