Java First-Tier: Aplicações

Tratamento de Exceções

Grupo de Linguagens de Programação

Departamento de Informática
PUC-Rio

Motivações para Exceções

- Um método pode detectar uma falha mas não estar apto a resolver sua causa, devendo repassar essa função a quem saiba
- Se introduzirmos o tratamento de falhas ao longo do fluxo normal de código, podemos estar comprometendo muito a inteligibilidade

2

Exceções

- Diz-se que uma exceção é *lançada* para sinalizar alguma falha
- O lançamento de uma exceção causa uma interrupção abrupta do trecho de código que a gerou
- O controle da execução volta para o primeiro trecho de código (na pilha de chamadas) apto a tratar a exceção lançada

3

Suporte a Exceções

- As linguagens OO tipicamente dão suporte ao uso de exceções
- Para usarmos exceções precisamos de:
  - uma representação para a exceção
  - uma forma de lançar a exceção
  - uma forma de tratar a exceção

4

Exceções em Java

- Java dá suporte ao uso de exceções:
  - são representadas por classes
  - são lançadas pelo comando **throw**
  - são tratadas pela estrutura **try-catch-finally**
- De modo geral, um método que lance uma exceção deve declarar isso explicitamente
- Para uma classe representar uma exceção, ela deve pertencer a uma certa hierarquia

5

Exemplo de Uso

- Considere a classe:

```
public class Calc {
  public int div(int a, int b) {
    return a/b;
  }
}
```

- O método **div**, se for chamado com **b** igual à zero, dará um erro. Esse erro poderia ser sinalizado através de uma exceção

6

Modelando uma Exceção

- Vamos, então, modelar uma exceção que indica uma tentativa de divisão por zero:

```
public class DivByZero extends Exception {
  public String toString() {
    return "Division by zero.";
  }
}
```

7

Lançando uma Exceção

- Agora vamos fazer com que o método **div** lance a exceção que criamos:

```
public class Calc {
  public int div(int a, int b) throws DivByZero {
    if (b == 0) throw new DivByZero();
    return a/b;
  }
}
```

8

Tratando uma Exceção

- Podemos, agora, utilizar o método **div** e tratar a exceção, caso esta ocorra:

```
...
Calc calc = new Calc();
try {
  int div = calc.div(x, y);
  System.out.println(div);
} catch (DivByZero e) {
  System.out.println(e);
}
...
```

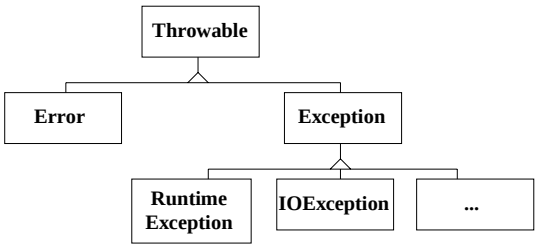
9

Tipos de Exceções em Java

- Java possui dois tipos de exceções:
  - ***Checked Exceptions*** são exceções que devem ser usadas para modelar falhas contornáveis. Devem sempre ser declaradas pelos métodos que as lançam e precisam ser tratadas (a menos que explicitamente passadas adiante)
  - ***Unchecked Exceptions*** são exceções que devem ser usadas para modelar falhas incontornáveis. Não precisam ser declaradas e nem tratadas

10

Hierarquia de Exceções



11

Checked Exceptions

- Para criarmos uma classe que modela uma *checked exception*, devemos estender a classe **Exception**
- Essa exceção será sempre verificada pelo compilador para garantir que seja tratada quando recebida e declarada pelos métodos que a lançam

12

Unchecked Exceptions

- Para criarmos uma classe que modela uma *unchecked exception*, devemos estender a classe **Error** ou **RuntimeException**
- Esse tipo de exceção não será verificado pelo compilador
- Tipicamente não criamos exceções desse tipo, elas são usadas pela própria linguagem para sinalizar condições de erro

13

Repassando Exceções

- Se quiséssemos usar o método **div** sem tratar a exceção, deveríamos declarar que a exceção deve ser passada adiante:

```
public void f() throws DivByZero {
  Calc calc = new Calc();
  int div = calc.div(a,b);
  System.out.println(div);
}
```

14

Tratando e Repassando Exceções

- Mesmo tratada, a exceção pode ser repassada:

```
public void f() throws DivByZero {
  Calc calc = new Calc();
  try {
    int div = calc.div(x, y);
    System.out.println(div);
  } catch (DivByZero e) {
    System.out.println(e);
    throw e;
  }
}
```

15

Estrutura **try-catch-finally**

Como apresentado, usamos **try-catch** para tratar uma exceção. A terceira parte dessa estrutura, **finally**, especifica um trecho de código que será *sempre* executado, não importando o que acontecer dentro do bloco **try-catch**

*Não é possível deixar um bloco **try-catch-finally** sem executar sua parte **finally***

16

Exemplo de **try-catch-finally**

```
void readFile(String name) throws IOException {
  FileReader file = null;
  try {
    file = new FileReader(name);
    ... // lê o arquivo
  } catch (Exception e) {
    System.out.println(e);
  } finally {
    if (file != null) file.close();
  }
}
```

17

Tratando Múltiplas Exceções

```
try {
  ...
} catch (Exception1 e1) {
  ...
} catch (Exception2 e2) {
  ...
} finally {
  ...
}
```

18