Programação I

Tratamento de Exceções

Samuel da Silva Feitosa

Aula 11

Tratamento de Erros (1)

- Como erros podem ocorrer durante a execução de uma aplicação, devemos definir como eles serão tratados.
  - Tradicionalmente, códigos de erro são utilizados para lidar com falhas na execução de um programa.
  - Nesta abordagem, os métodos devolveriam números inteiros para indicar o tipo de erro que ocorreu.

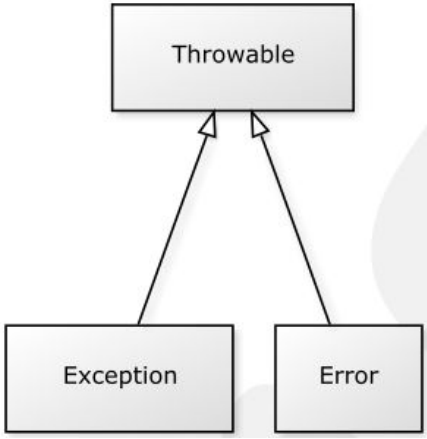
```
public int deposita(double valor) {
    if (valor < 0) {
        return 107; // código de erro para valor negativo
    }

    this.saldo += valor ;
    return 0; // sucesso
}
```

Tratamento de Erros (2)

- Utilizar códigos de erro exige uma vasta documentação dos métodos para explicar o que cada código significa.
  - Além disso, esta abordagem “gasta” o retorno do método impossibilitando que outros tipos de dados sejam devolvidos.
  - Em outras palavras, ou utilizamos o retorno para devolver códigos de erro ou para devolver algo pertinente a lógica natural do método.
  - Não é possível fazer as duas coisas sem nenhum tipo de “gambiarra”.
- Observe que no código do método geraRelatorio() seria necessário devolver dois tipos de dados incompatíveis: int e referências de objetos da classe Relatorio.

```
??? geraRelatorio() {
  if(...) {
    return 200; // código de erro tipo1
  } else {
    Relatorio relatorio = ...
    return relatorio;
  }
}
```

Throwable, Exception e Error



- Throwable
  - Representa a superclasse de todos os elementos sinalizadores de exceções.
  - Apenas objetos dessa classe podem ser lançados pela JVM com o uso da diretiva **throw**.
- Exception (extends Throwable)
  - Dá origem a um conjunto de subclasses que indica condições anormais em uma aplicação.
- Error (extends Throwable)
  - Dá origem a um conjunto restrito de subclasses que indicam condições de erros severas, em geral associadas a JVM ou ao sistema em si.

Tipos de Exceções

- Não monitoradas (RuntimeException)
  - Não exigem o tratamento com o uso de diretivas **try/catch/finally**, passando implicitamente para o contexto superior.
  - São subclasses de java.lang.RuntimeException.
- Monitoradas (Exception)
  - Exigem tratamento obrigatório com **try/catch/finally** ou declaração explícita de seu lançamento para contexto superior por meio da cláusula **throws**.
  - É o compilador que verifica e exige o tratamento ou a declaração explícita do lançamento de exceções.
  - São subclasses da classe java.lang.Exception.

Lançando uma Exceção não monitorada

- Quando identificamos um erro, podemos criar um objeto de alguma Exceção não monitorada e “lançar” a referência dele com o comando throw.

```
public void deposita(double valor) {
    if (valor < 0) {
        throw new RuntimeException("Valor negativo inválido!");
    }

    this.saldo += valor ;
}
```

Lançando uma Exceção monitorada

- Quando identificamos um erro, podemos criar um objeto de alguma Exceção monitorada e “lançar” a referência dele com o comando throw.
  - Contudo, antes de lançar uma Exceção monitorada, é necessário determinar de maneira explícita através do comando throws que o método pode lançar esse tipo de erro.

```
public void deposita(double valor) throws Exception {
    if (valor < 0) {
        throw new Exception("Valor negativo inválido!");
    }

    this.saldo += valor ;
}
```

Capturando Exceções (1)

- Quando estamos utilizando as classes que lançam exceções, é possível capturá-las de forma a tratar os erros usando os comandos **try/catch**.
  - O tratamento de exceções funciona da mesma forma para as monitoradas e não monitoradas.
  - A diferença é que as exceções monitoradas precisam ser obrigatoriamente tratadas.

```
ContaPoupanca cp = new ContaPoupanca();

try {
    cp.deposita(100);
} catch (Exception e) {
    System.out.println("Erro ao depositar: " + e.getMessage());
}
```

Capturando Exceções (2)

- É possível encadear vários blocos catch para capturar exceptions de classes diferentes.

```
ContaPoupanca cp = new ContaPoupanca();

try {
    cp.deposita(100);
} catch (IllegalArgumentException e) {
    System.out.println("Argumento inválido!");
} catch ( SQLException e ) {
    System.out.println("Erro ao salvar!");
} catch (RuntimeException e) {
    System.out.println("Erro desconhecido ao depositar!");
}
```

Capturando Exceções (3)

- Se um erro acontecer no bloco try ele é abortado.
  - Consequentemente, nem sempre todas as linhas do bloco try serão executadas.
  - Além disso, somente um bloco catch é executado quando ocorre um erro.
- Em alguns casos, queremos executar um trecho de código independentemente se houver erros ou não.
  - Para isso podemos, utilizar o bloco **finally**.

```
try {
    // Código
} catch (ArithmeticException e) {
    System.out.println("Tratamento de ArithmeticException");
} catch (Exception e) {
    System.out.println("Tratamento de Exception");
} finally {
    // código que deve ser sempre executado
}
```

Exercício - OOP2

- Para implementar jogos com cartas são necessárias classes que representam uma carta individual e também um baralho. Implemente essas classes, considerando que as Cartas são Ás, 2 a 10, valete, dama e rei, e os naipes são copas, espadas, ouros e paus; e Baralho é um conjunto de 52 cartas (13 cartas para cada naipe), em ordem ou embaralhado.
  - Faça os devidos tratamentos de exceções para as classes Cartas e Baralho.
  - Implemente um método para embaralhar as cartas de um baralho utilizando como recurso apenas um gerador de números aleatórios.
- A entrega desta atividade será via Moodle.
  - Enviar um arquivo zip apenas os códigos Java desenvolvidos no exercício.