Programação Orientada a Objetos
SANTOS, Rafael (PLT)

Capítulo 6 – Estruturas de decisão/seleção (condicionais)

Atribuição (=)

- Possibilita atribuições em **sequência** na mesma instrução
- **Retorna** o valor atribuído
- Atribuições compostas: **incluem a própria variável**: +=, *=, -=, etc

Operadores

- **Aritméticos**: *, /, %, +, -
- **Relacionais**: ==, !=, <=, >=, <, >
- Lógicos **abreviados**: &&(e), ||(ou)
- Lógicos **não abreviados**: &(e), |(ou), ^(ou exclusivo)
- Lógico **unário**: !(não ou inversão)
- **Bitwise**: |(ou), &(e), ^(ou exclusivo) e ~(não)
- Ver classes:
  - Comparavel (Cap. 6)
  - DemoComparavel (Cap. 6)

Abreviados X não abreviados

```
public static void main(String[] args) {

        int z = 5;
        if (++z > 5 || ++z > 6) {
            z++;
        }

        int y = 5;
        if (++y > 5 | ++y > 6) {
            y++;
        }

        System.out.println("valor de z: " + z);
        System.out.println("valor de y: " + y);

    }
```

Instrução if-else

- Tem uma c**ondição** que deve estar entre **parênteses**
- Se o bloco que será executado contiver apenas **uma instrução**, as **chaves não** são **obrigatórias**, mas **recomendadas**
- Podem ser **aninhados**
- Ver classes:
  - EntradaDeCinema (Cap. 6)
  - DataIf (Cap. 6)

Ifs aninhados

- O palavra-chave ***else*** sempre se refere ao ***if*** aberto
- Caso não haja um ***if*** aberto para um ***else***, ocorrerá erro de compilação
- Sempre use bloco delimitados por chaves tanto para o ***if*** como para o ***else***

Operador ternário ?

- Trata-se de um **if-else** com uma **sintaxe abreviada**, mais adequada à composição de **expressões**
- Ver classe:
  - ComparaSimples (Cap. 6)

Instrução switch

- Usada, no lugar de um conjunto de estruturas if-else, para avaliar dados dos tipos: **byte**, **short**, **int**, **char** e **enum**
- Ver classe:
  - DataSwitch (Cap. 6)

Enums

- **Tipo**, definido como uma classe, que **contém constantes** (enums) com características de **objetos**, declarada em um **arquivo próprio** ou em uma classe
- Pode conter:
  - Construtores, inclusive sobrecarregados
  - Variáveis
  - Métodos
  - Corpo de classe específico de constantes (parece a implementação de classe anônima)

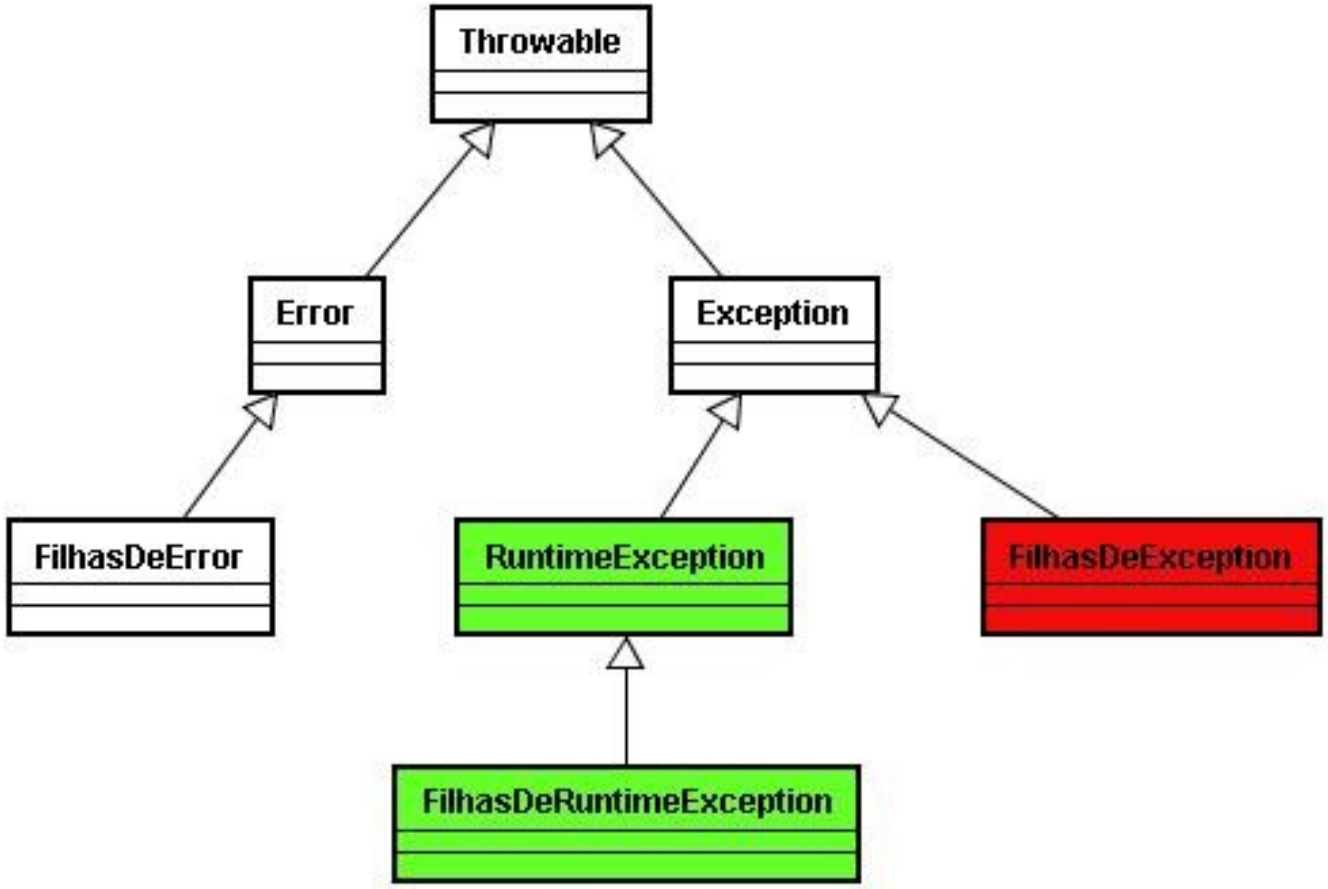
Exercício

- Crie enums para representar o seguinte:
  - Estado civil
  - Situação funcional
  - Grau de instrução
  - Tipo de imóvel
  - Tipo de veículo
- Para cada um dos exemplo, crie uma classe de teste que use uma instrução switch-case

Tratamento de exceções

- **Exceções** são representadas por **classes**, pertencentes a uma **hierarquia própria**
- Podem ser **tratadas** ou **passadas** ao método chamador
- Para tratamento, usamos **try-catch-finaly**
- Para passagem, usamos **throws**
- Podem ser geradas pela **JVM** ou pelo **aplicativo**
- Para **lançar** uma exceção, usamos **throw**

Hierarquia das exceções



Bloco try-catch-finally

- Permite alterar o **fluxo normal** do programa e chamar um **manipulador de exceção**, quando uma condição excepcional ocorrer
- O bloco de **try** define uma **região protegida** ou de risco
- Um bloco **catch** define um **manipulador de exceção**, em ordem, **após** o bloco try
- O bloco **finally** será **sempre** executado
- **Pelo menos um** bloco catch ou finally **deve** ser especificado depois do bloco try

Exemplo

```
public class TestaTryCatchFinally {

    public static void main(String[] args) {

        Object o = "teste";
        try {
            System.out.println(o.toString());
            System.out.println(“Sucesso!!!");
        } catch (NullPointerException e) {
            System.out.println("Erro de objeto nulo!!!");
        } finally {
            System.out.println("Mensagem de finally!!!");
        }
    }
}
```

Exceção verificada capturada

```
package testesComExcecoes;

public class TestaExcecaoVerificada {

    public static void main(String[] args) {

        try {
            Class.forName("pacote.NomeDaClasse");
            System.out.println("Sucesso!!!");
        } catch (ClassNotFoundException ex) {
            System.out.println("Nao encontrada!!!");
        }

    }
}
```

Exceção verificada declarada

```
package testesComExcecoes;

public class TestaExcecaoVerificadaDeclarada {

    public static void main(String[] args)
            throws ClassNotFoundException {

        Class.forName(“pacote.ClasseASerCarregada");
        System.out.println("Sucesso!!!");

    }

}
```

Um declara, outro captura

```
public class TestaExcecaoVerificadaDeclarada {

    public static void main(String[] args) {
        try {
            carrega(“pacote.ClasseASerCarragada");
        } catch (ClassNotFoundException e) {
            System.out.println("Erro de carga!!!");
        }
    }

    static void carrega(String classe)
            throws ClassNotFoundException {
        Class.forName(classe);
        System.out.println("Sucesso!!!");
    }

}
```

Criação de exceções

- Cria-se um classe que **estende** uma classe da hierarquia das exceções, mais comumente **Exception**

```
package testesComExcecoes;

public class NumeroForaDoIntervaloExcecao
        extends Exception {

}
```

Lançamento de exceções

```
public static void main(String[] args) {
    try {
        recebeNumero(100);
        System.out.println("100 recebido");
        recebeNumero(120);
        System.out.println("120 recebido");
    } catch (NumeroForaDoIntervaloExcecao e) {
        System.out.println(
                "Erro de recebimento");
    }
}
static void recebeNumero(int a)
        throws NumeroForaDoIntervaloExcecao {
    if (a < 0 || a > 100) {
        throw new NumeroForaDoIntervaloExcecao();
    }
}
```

Exercício

- Crie uma classe que contenha um método que receba, pelo menos, dois parâmetros e lance, quando necessário, uma exceção para cada um deles, indicando possíveis condições de erro
- Crie as exceções e uma classe de teste para executar, a partir de main, o método desenvolvido