POO 2

Msc. Édimo Sousa Silva.

Sistemas de Informação

Faculdade Paraíso (FAP)

Aula 02 -
Typecast, Generics e enum

- **Typecast (Coersão)**
- **Typecast (Conversão explicita)**
- **Typecast (Upcasting)**
- **Typecast (Downcasting)**
- **Instanceof**
- **getClasse**
- **Generics**
- **Enum**

Typecast - Coersão (Dados primitivos)

É feito automaticamente pelo compilador quando um tipo de dado pode ser facilmente manipulado no lugar de outro tipo de dado.

```
private static void typecastCoersao() {

        int numInt = Integer.MAX_VALUE;

        long numLong = Long.MAX_VALUE;

        System.out.println("int = " + numInt);

        System.out.println("long = " + numLong);

        numLong = numInt;

        System.out.println("long apos receber inteiro = " + numLong + "\n");

}
```

Typecast - Conversão explicita Dados Primitivos

Quando o código indica ao compilador o tipo que quer converter. Geralmente é usado quando a conversão pode apresentar muita perda de informação (por exemplo, ao converter de long para short);

```
private static void typecastConversaoExplicita() {

        int numInt = Integer.MAX_VALUE;

        long numLong = 0;

        System.out.println("int = " + numInt);

        System.out.println("long = " + numLong);

        numInt = (int) numLong;

        System.out.println("int apos receber long = " + numInt + "\n");

}
```

Typecast - Upcasting

Tipo de coerção quando se atribui um objeto de uma subclasse a uma superclasse. Essa atribuição é sempre válida.

```
private static void typecastObjetos() {

        Object obj = new TV(29, 1, 0, false);

        System.out.println("A variável obj é " + obj.getClass());

}
```

Downcasting

Ocorre quando o objeto se passa como se fosse um subtipo dele. Não há garantias que funcione (pode lançar uma ClassCastException, o que obviamente é um erro de programação) e pode haver necessidade de conversões.

```
private static void typecastDowncasting() {

                Object obj = new TV(29, 1, 0, false);

                TV tv = (TV) obj;

                TV tv2 = new TV(29, 1, 0, false);

                System.out.println("var tv = " + tv);

                System.out.println("var obj = " + obj);

                System.out.println("var tv2 = " + tv2);        }
```

Instanceof

Usado para verificar se o objeto é uma instancia de determinada classe

```
Object obj = new TV(29, 1, 0, false);
Object obj2 = "Uma string qualquer";
TV tv = null;
TV tv2 = null;
if (obj instanceof TV) {tv = (TV) obj;}
if (obj2 instanceof TV) {tv = (TV) obj;}
System.out.println("var tv = " + tv);
System.out.println("var tv2= " + tv2);
```

Typecast - getClass

Ocorre quando o objeto se passa como se fosse um subtipo dele. Não há garantias que funcione (pode lançar uma ClassCastException, o que obviamente é um erro de programação) e pode haver necessidade de conversões.

```
private static void typecastDowncasting() {

                Object obj = new TV(29, 1, 0, false);

                TV tv = (TV) obj;

                TV tv2 = new TV(29, 1, 0, false);

                System.out.println("var tv = " + tv);

                System.out.println("var obj = " + obj);

                System.out.println("var tv2 = " + tv2);        }
```

Generics

- **Generics, ou programação genérica, serve para determinar para o compilador, qual tipo de classe deve ser interpretada.**
- **Independente de qual classe seja, queremos trata-la da mesma forma.**
- **Sem a necessidade de usar recursos como herança ou a interface.**

Generics - Classe generica

```
public class Caixa<T> {
        T objeto;
        public Caixa(T objeto) {
                this.objeto = objeto;
        }
        public T getObjeto() {
                return objeto;
        }
        public void setObjeto(T objeto) {
                this.objeto = objeto;
        }
}
```

Generics - instanciando a classe

```
Caixa<TV> caixa1 = new Caixa<TV>(new TV(10, 20, 30, true));

Caixa<Radio> caixa2 = new Caixa<Radio>(new Radio(9.9f, " povo"));

TV tv1 = (TV) caixa1.getObjeto();

Radio radio1 = (Radio) caixa2.getObjeto();

System.out.println(tv1.ligada);

System.out.println(radio1.nome);
```

Enum - caracteristicas 1

- As instâncias dos tipos enum são criadas e nomeadas junto com a declaração da classe, sendo fixas e imutáveis (o valor é fixo);
- Não é permitido criar novas instâncias com a palavra chave new;
- O construtor é declarado private, embora não precise de modificador private explícito;

Enum - caracteristicas 2

- Seguindo a convenção, por serem objetos constantes e imutáveis (static final), os nomes declarados recebem todas as letras em MAIÚSCULAS;
- Opcionalmente, a declaração da classe pode incluir variáveis de instância, construtor, métodos de instância, de classe, etc.
- Usar enums aumenta a coesão do código.

Enum - criação

```
public enum TIPORADIO {
        AM("Amazing music"), FM("Fantastic music"), CM("Crying music");
        private final String valor;

        private TIPORADIO(String valor) {
                this.valor = valor;
        }

        public String getValor() {
                return this.valor;
        }
}
```

Enum - criação

```
public enum TIPORADIO {
        AM("Amazing music"), FM("Fantastic music"), CM("Crying music");
        private final String valor;
        private TIPORADIO(String valor) {
                this.valor = valor;
        }
        public String getValor() {
                return this.valor;
        }
}
```

Enum - usando

```
public enum TIPORADIO {
        AM("Amazing music"), FM("Fantastic music"), CM("Crying music");
        private final String valor;
        private TIPORADIO(String valor) {
                this.valor = valor;
        }
        public String getValor() {
                return this.valor;
        }
}
```

Enum - criação

```
System.out.println(radio1.nome);
TIPORADIO tr1 = TIPORADIO.AM;
System.out.println(tr1);
System.out.println(tr1.getValor());
TIPORADIO tr2 = TIPORADIO.AM;
System.out.println(tr2);
System.out.println(tr2.getValor());
TIPORADIO tr3 = TIPORADIO.CM;
System.out.println(tr3);
System.out.println(tr3.getValor());
System.out.println(tr1==tr3);
```

Bibliografia

- P. Boratti. Programação Orientada a Objetos em Java. 2007.
- P. Deitel, H. Deitel. Java Como Programar. 2010.