Polimorfismo

Prof. Edwar Saliba Júnior
Fevereiro de 2011

Tipos

- Verificação de Tipos:
  - Atividade que garante que os operandos utilizados com um operador sejam de tipos compatíveis;
- Tipo Compatível:
  - É um tipo cujo os valores são adequados a realização da operação designada pelo operador ou que pode ser convertido implicitamente em um tipo cujos valores sejam adequados.

Polimorfismo

- Característica que possibilita a criação de código capaz de operar sobre valores distintos;
- Resumindo: onde se espera um valor do tipo X pode-se receber um valor do tipo Y;
- Polimorfismo = “Muitas formas”.

Coersão

- É a conversão implícita de tipos;
- Quando uma operação é realizada sobre um tipo diferente do esperado, o compilador verifica se é possível fazer realizar a conversão;
- Exemplo:
  - Na linguagem C é possível atribuir um valor do tipo `char` a uma variável do tipo `int`.

```
int a = ‘r’;
```

Sobrecarga ou *Overloading*

- Um identificador ou operador é sobrecarregado quando pode ser utilizado para designar duas ou mais operações distintas;
- Exemplo:
  - O operador + em Java pode ser utilizado tanto para somar valores quanto para concatenar *strings*.

```
int res = 5 + 3.14;
System.out.printf(“Total = ” + res);
```

Inclusão

- Um subtipo S de um tipo T é formado pelo subconjunto dos valores de T. Assim, todo valor de S deve ser também um valor de T;
- Exemplo:
  - Considere as classes A e B mostradas a seguir. A herança apresentada também define uma situação polimórfica.

Exemplo de Polimorfismo

```java
package polimorfismo;

public class A {
    protected int x, y;
    private int z;

    public A (int a, int b, int c){
        this.x = a;
        this.y = b;
        this.z = c;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getZ() {
        return z;
    }

    public void imprimeValores(){
        System.out.println("Valor de X: " + x);
        System.out.println("Valor de Y: " + y);
        System.out.println("Valor de Z: " + z);
    }
}
```

```java
package polimorfismo;

public class B extends A {

    private int k;

    public B(int a, int b, int c, int d){
        super(a,b,c);
        this.k = d;
    }

    @Override
    public void imprimeValores(){
        System.out.println("Valor de X: " + x);
        System.out.println("Valor de Y: " + y);
        System.out.println("Valor de Z: " + getZ());
        System.out.println("Valor de K: " + k);
    }
}
```

Exemplo de Polimorfismo

- Pode-se verificar que o método “testePolimorfismo” espera receber como parâmetro um objeto do tipo A. Contudo, são passados para o método tanto objetos do tipo A quanto objetos do tipo B e ainda assim o método funciona normalmente.

```
1   package polimorfismo;
2
3   public class Main {
4
5       public static void main(String[] args) {
6           B obj1 = new B(10,20,30,40);
7           A obj2 = new A(200,300,400);
8
9           testePolimorfismo(obj2);
10          testePolimorfismo(obj1);
11      }
12
13      public static void testePolimorfismo(A obj){
14          obj.imprimeValores();
15      }
16  }
```

```
Output - Polimorfismo (run)
run:
Valor de X: 200
Valor de Y: 300
Valor de Z: 400
Valor de X: 10
Valor de Y: 20
Valor de Z: 30
Valor de K: 40
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operador `instanceOf`

- O operadora `instanceOf` serve para verificarmos se um objeto é uma instância de uma determinada classe;

- Sintaxe:

```
obj instanceOf classe
```

- Se o `obj` for uma instância da `classe`, então está operação retornará `true`, caso contrário retornará `false`.

Método `getClass()`

- O método `getClass()` retorna um objeto `Class` que representa a instância da classe deste objeto;
- E para sabermos o nome classe do objeto, basta utilizarmos o método `getName()` do objeto retornado por `getClass()`;
- Sintaxe:

```
obj.getClass().getName();
```

Downcast

- *Downcast* é uma conversão explícita de um objeto para sua própria classe ou para uma de suas superclasses;
- Exemplo: *downcast* de um objeto da classe Funcionário para um objeto da sua superclasse Pessoa;

```
Funcionario f;

Pessoa p;

p = (Pessoa) f;
```

Downcast

- O *downcast* só é possível para um objeto em seu próprio tipo ou no tipo de uma de suas superclasses;

- Essa verificação ocorre em tempo de execução. Caso a conversão não seja do tipo ou supertipo do objeto, ocorrerá um exceção: `ClassCastException`.

Exemplo 02

- Em uma aplicação existem as seguintes classes: Pessoa, Cliente e Funcionário. Deseja-se criar uma lista de Pessoas;

- Neste exemplo utilizaremos uma coleção `ArrayList` para armazenarmos os dados;

- Exemplo: .pdf

Bibliografia

- DEITEL, H. M.; DEITEL, P. J. **Java Como Programar**; tradução Edson Furmankiewicz; revisão técnica Fábio Lucchini. 6a. ed., São Paulo: Pearson, 2005.
- FERREIRA, Kecia Aline Marques. *Slides* da disciplina de Programação de Computadores II. CEFET-MG, 2009.