Polimorfismo

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior
Janeiro de 2020

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”.

Coerção

- É a conversão implícita de tipos;
- Quando uma operação é realizada sobre um tipo diferente do esperado, o compilador verifica se é possível fazer a conversão forçada (sem *typecast*);
- 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 no próximo *slide*. 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 operador **instanceOf** serve para verificarmos se um objeto é uma instância de uma determinada classe;

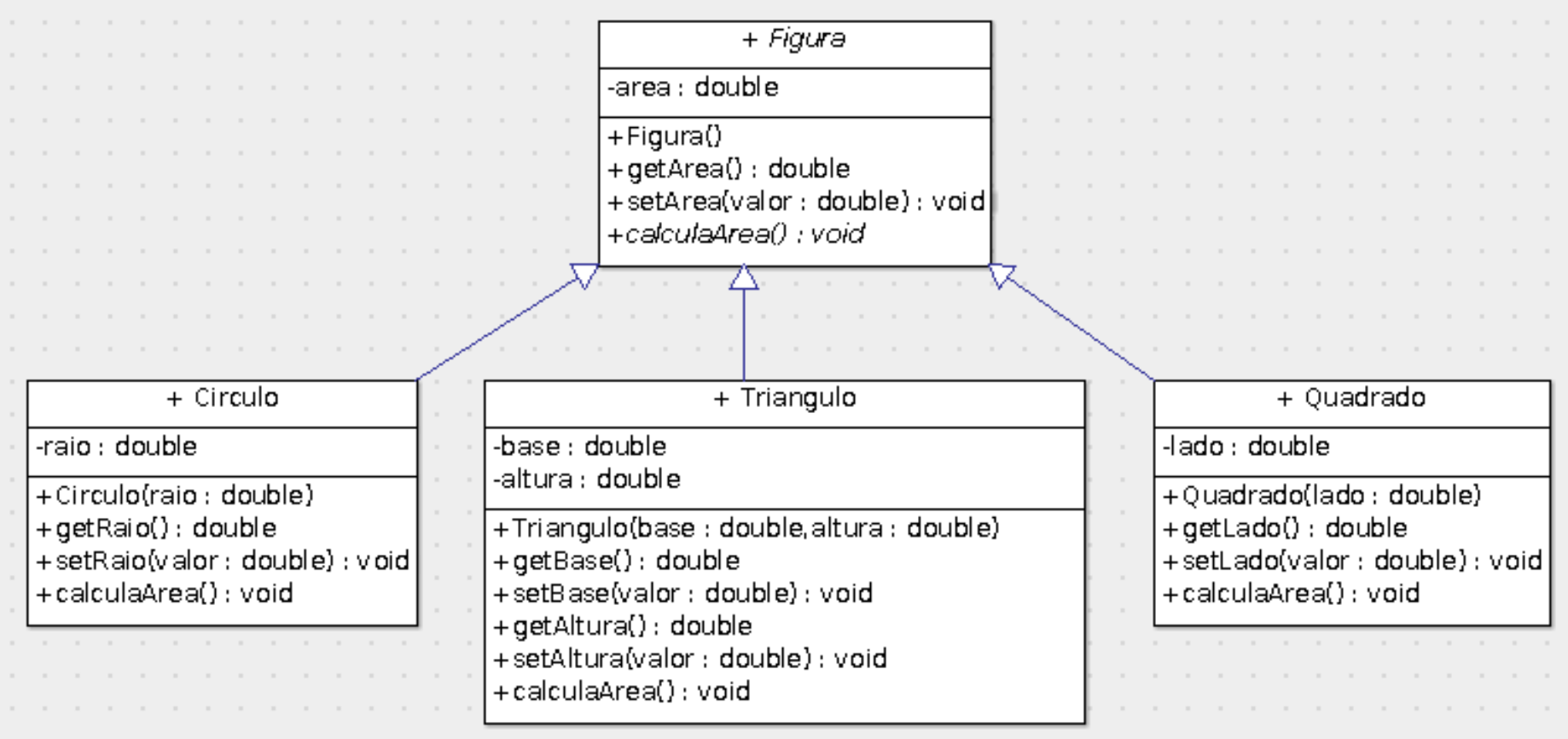
- Sintaxe:

```
obj instanceOf classe
```

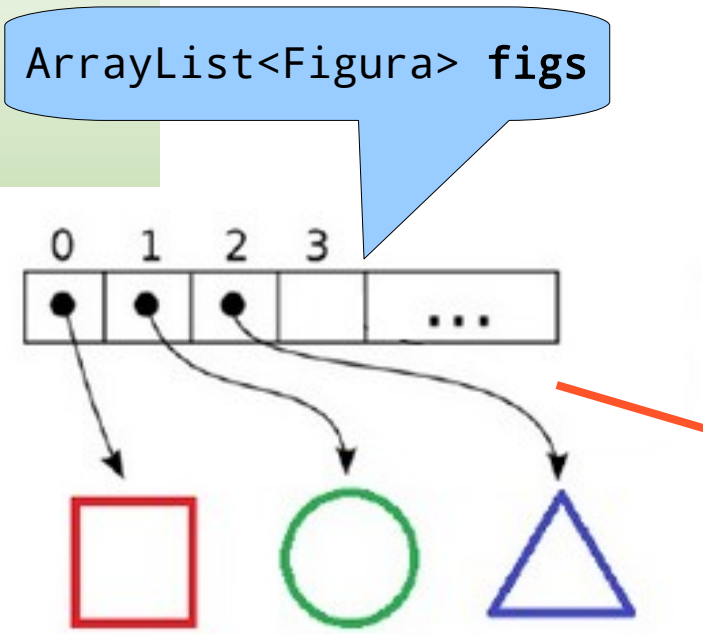
- Se **obj** for uma instância da classe, então esta operação retornará **true**, caso contrário retornará **false**.

Polimorfismo

Considere que o código-fonte das classes deste diagrama foi criado.



Polimorfismo




```java
package principal;

import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {
        ArrayList<Figura> figs = new ArrayList<>();
        Scanner sc = new Scanner(System.in);

        Quadrado quadrado = new Quadrado(15);
        Circulo circulo = new Circulo(20);
        Triangulo triangulo = new Triangulo(25,10);

        figs.add(quadrado);
        figs.add(circulo);
        figs.add(triangulo);

        // Testa se a figura "triangulo" está cadastrada no ArrayList
        if(contem(triangulo,figs))
            System.out.println("Existe um triângulo cadastrado.");

        // Mostra todas as figuras cadastradas no ArrayList
        for(Figura f : figs) {
            mostrar(f);
            System.out.println("-x-x-x-x-");
        }

        sc.close();
    }
```

Polimorfismo

- Função polimórfica

```
33    public static void mostrar(Figura v) {
34        if(v instanceof Quadrado)
35            System.out.println(" - Quadrado: ");
36        else
37            if(v instanceof Triangulo)
38                System.out.println(" - Triângulo: ");
39            else
40                System.out.println(" - Círculo: ");
41
42        if(v instanceof Quadrado) {
43            Quadrado q = (Quadrado)v;
44            System.out.println(" - lado: " + q.getLado());
45            q.calculaArea();
46        }
47        else {
48            if(v instanceof Triangulo) {
49                Triangulo t = (Triangulo)v;
50                System.out.println(" - base: " + t.getBase());
51                System.out.println(" - altura: " + t.getAltura());
52                t.calculaArea();
53            }
54            else {
55                Circulo c = (Circulo)v;
56                System.out.println(" - raio: " + c.getRaio());
57                c.calculaArea();
58            }
59        }
60        System.out.println(" - área : " + v.getArea());
61    }
```

Polimorfismo

- Outra função polimórfica

```
63    public static boolean contem(Figura fig, ArrayList<Figura> figuras) {
64        int i = 0;
65        boolean achou = false;
66
67        while((i < figuras.size()) && (!achou)) {
68            if((fig instanceof Quadrado) && (figuras.get(i) instanceof Quadrado)) {
69                Quadrado q = (Quadrado)fig;
70                Quadrado q2 = (Quadrado)figuras.get(i);
71                achou = q.equals(q2);
72            }
73            else {
74                if((fig instanceof Triangulo) && (figuras.get(i) instanceof Triangulo)) {
75                    Triangulo t = (Triangulo)fig;
76                    Triangulo t2 = (Triangulo)figuras.get(i);
77                    achou = t.equals(t2);
78                }
79                else {
80                    if((fig instanceof Circulo) && (figuras.get(i) instanceof Circulo)) {
81                        Circulo c = (Circulo)fig;
82                        Circulo c2 = (Circulo)figuras.get(i);
83                        achou = c.equals(c2);
84                    }
85                }
86            }
87
88            i++;
89        }
90
91        return achou;
92    }
93 }
```

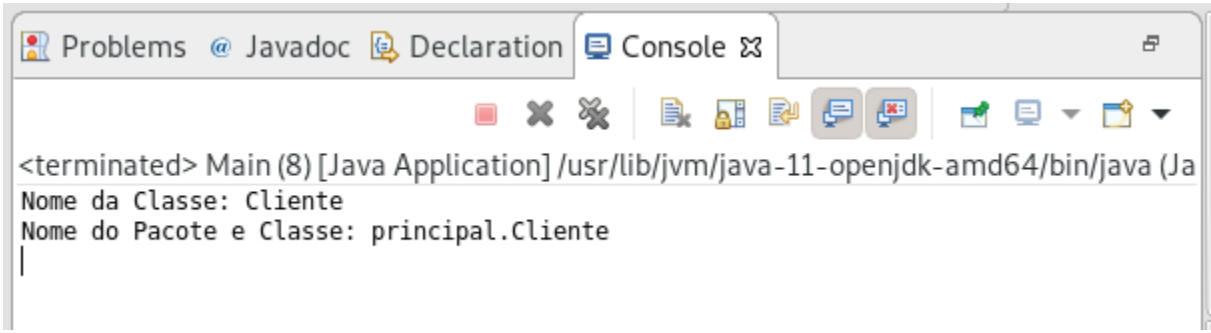
Método getClass()

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

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

Método getClass()

```java
package principal;

class Cliente{
    private String nome;
    private String telefone;

    public Cliente(String nome, String tel) {
        this.nome = nome;
        this.telefone = tel;
    }

    public String getNome() {
        return nome;
    }

    public String getTelefone() {
        return telefone;
    }
}


public class Main {

    public static void main(String[] args) {
        Cliente cli = new Cliente("Edwar Saliba Júnior","(31)99999-8888");

        String nomeDaClasse = cli.getClass().getSimpleName();
        String nomeDoPacoteClasse = cli.getClass().getName();

        System.out.println("Nome da Classe: " + nomeDaClasse);
        System.out.println("Nome do Pacote e Classe: " + nomeDoPacoteClasse);
    }
}
```



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.