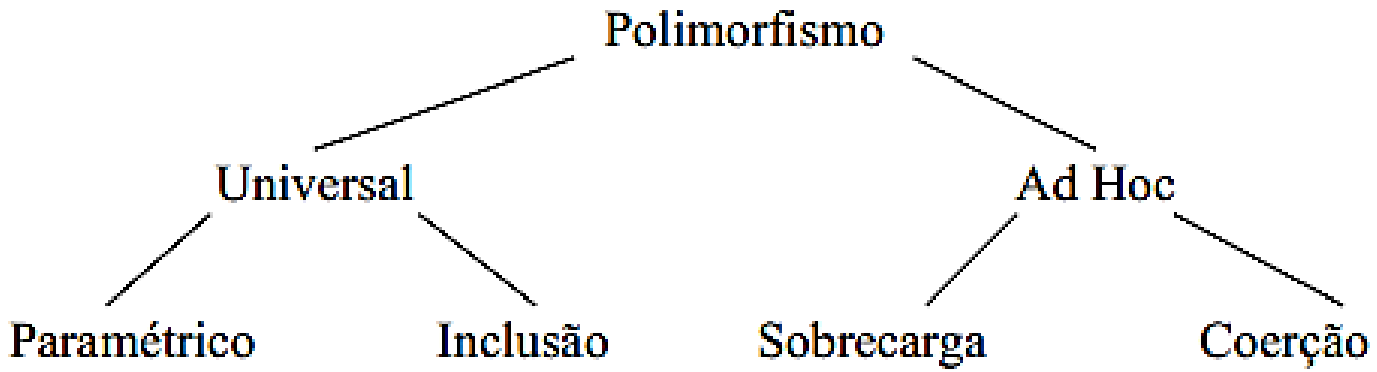
POO: Polimofismo

Fabio Lubacheski
fabio.lubacheski@mackenzie.br

Polimorfismo

- **Polimorfismo** é a capacidade de um objeto poder ser referenciado de várias formas, mas cuidado, polimorfismo **não quer dizer que o objeto fica se transformando**, muito pelo contrário, um objeto nasce de um tipo e morre daquele tipo, o que pode mudar é a maneira como nos referimos a ele.



Polimorfismo

- **Ad Hoc** – é caracterizado pela ausência de um modo uniforme no comportamento de um método, que pode variar radicalmente, de acordo com os tipos de seus argumentos de entrada.
  - **Sobrecarga** (*overloading*): permite que um nome de método seja usado mais do que uma vez, com **diferentes tipos de parâmetros**.

```
public class A{
  int metodo() { return 1; };
  int metodo( int x ) { return 1+x; };
}
```

  - **Coerção (*casting*)**: proporciona um meio de contornar a rigidez de tipos monofórmicos. Realizam um **mapeamento interno entre tipos**, de acordo com relações de equivalência existentes entre eles.

```
public boolean equals(Object obj){
        Ponto p=(Ponto)obj;
        . . . . . .
```

Polimorfismo

- **Universal** – também conhecido como polimorfismo verdadeiro, uma função ou tipo trabalha uniformemente para qualquer um dos tipos que ele seja compatível.
  - **Paramétrico**: possibilita que um único método possa ser codificado, de modo a funcionar uniformemente com vários tipos distintos. Também são **chamadas de funções genéricas**.

    Exemplo:

    ```
    ArrayList<Ponto> pontos=new ArrayList<Ponto>();
    ```

    Este tipo de declaração diz que cada elemento de `ArrayList` será do tipo `Ponto`, pois a classe `ArrayList` é um ***container*** de objetos que podem ser parametizados.

Polimorfismo

- **Universal**.
  - **Inclusão (override ou sobrescrita)**: é uma consequência do fato das subclasses (ou subtipos) herdarem automaticamente todas as operações das suas respectivas superclasses (ou supertipos), esse tipo de polimorfismo permite **sobrescrever os métodos herdados**.

```
public class Classe1{
  int Calcula() { return 1; };
}
public class Classe2 extends Classe1 {
  @Override
  int Calcula() { return 2; }
}
```

Polimorfismo e Herança

- Com o Polimorfismo e a Herança podemos estender uma classe a partir de outra e reescrever os métodos que acharmos necessário, por exemplo:

```
public class Classe1{
  int Calcula() { return 1; };
  String Metodo1() { return "metodo 1";}
}
public class Classe2 extends Classe1 {
  @Override
  int Calcula() { return 2; }
  String Metodo2() { return "metodo 2";}
}
```

Polimorfismo e Herança

- Note que os métodos `Metodo1()` não foi reescrito na `Classe2` além disso na `Classe2` foi acrescentado o método `Metodo2()`.

```
public class Classe1{
  int Calcula() { return 1; };
  String Metodo1() { return "metodo 1";}
}
public class Classe2 extends Classe1 {
  @Override
  int Calcula() { return 2; }
  String Metodo2() { return "metodo 2";}
}
```

Polimorfismo e Herança

- Com isso podemos ter o seguinte trecho de código

```
Classe1 c1 = new Classe1();
Classe2 c2 = new Classe2();
System.out.println(c1.Metodo1());
System.out.println(c2.Metodo2());
```

- E podemos ter também o seguinte trecho:

```
Classe2 c2 = new Classe2();
System.out.println(c2.Metodo1());
```

- Isto só é possível porque `Classe2` é um subtipo da `Classe1`.

Polimorfismo e Herança

- Porém, se invocarmos o método `Metodo2()` diretamente de `c1`, receberemos um erro de compilação, pois o `Metodo2()` não está definido na `Classe1`.

```
Classe1 c1 = new Classe2();
System.out.println(c1.Metodo2());
```

```
testaClasses.java:4: error: cannot find symbol
      System.out.println(c1.Metodo2());
                                ^
  symbol:   method Metodo2()
  location: variable c1 of type Classe1
```

Polimorfismo e Herança

- Para contornar isso podemos fazer uma coerção no objeto `c1`, quando necessário, para poder invocar o `metodo2()`.

```
Classe1 c1 = new Classe2();
System.out.println(((Classe2)c1).Metodo2());
```

- Agora dado que podemos atribuir a uma variável de referência da classe base (pai) tanto um objeto da classe derivada (filha), como saber de qual classe é determinado objeto ?

Polimorfismo e Herança

- Para contornar isto, podemos verificar dinamicamente qual o objeto está armazenado em uma variável de referência e, se necessário, realizar uma **coerção** (*casting*) para invocar o método corretamente:

```
if( c1 instanceof Classe2 )
  System.out.println(((Classe2)c1).Metodo2());
else
  System.out.println(c1.Metodo1());
```

- A palavra reservada **`instanceof`** verifica se um determinado objeto é instância de uma determinada classe.

Polimorfismo e Herança

- Importante, uma variável de referência da classe derivada (filha) não suporta um objeto da classe base (pai), por exemplo:

```
Classe2 c2 = new Classe1();
```

- Dado o que vimos até agora, se tivermos um polimorfismo de inclusão (*override* ou sobrescrita) entre os métodos das classes 1 e 2 e fizéssemos uma chamada conforme abaixo, qual seria método invocado ?

```
Classe1 c1 = new Classe2();
System.out.println(c1.Calcula());
```

Polimorfismo e Herança

- Sempre é chamado o método sobrescrito do objeto instanciado, no exemplo é chamado o método calcula da `Classe2`, mesmo fazendo coerção, pois o método `Calcula()` foi reescrito na `Classe2`.

```
Classe1 c1 = new Classe2();
System.out.println(c1.Calcula());
>>> saída: 2
```

- Isso acontece porque na sobrescrita de métodos, a busca da implementação do método é executado de baixo para cima na hierarquia

Exercício

Considere as seguintes classes descritas abaixo.

```
public class Classe1{
    int Calcula(){ return 1; };
}
public class Classe2 extends Classe1{
    int Calcula(){ return super.Calcula(); }
}
public class Classe3 extends Classe2{
    int Calcula(){ return 3; }
}
```

Exercício

Se as classes forem utilizadas a partir do programa a seguir, qual seria a saída do programa ?

```
public static void main(String[] args) {
    int Result=0;
    Classe1 Objs[] = new Classe1[3];
    Objs[0] = new Classe1();
    Objs[1] = new Classe2();
    Objs[2] = new Classe3();
    for (int i=0; i<3; i++)
        Result += Objs[i].Calcula();
    System.out.println( Result );
}
```

Fim