Polimorfismo

PROF.ME. HÉLIO ESPERIDIÃO

Polimorfismo

Traduzindo, do grego, ao pé da letra, polimorfismo significa "muitas formas".

Essas formas, em nosso contexto de programação, são as subclasses/objetos criados a partir de uma classe maior, mais geral, ou abstrata

Polimorfismo é a capacidade da orientação a objeto que permite controlar todas as formas de uma maneira mais simples e geral, sem ter que se preocupar com cada objeto especificamente.

O polimorfismo ad hoc

O termo *ad hoc* tem origem no latim e significa "*para esta finalidade*".

O termo é usado para tudo aquilo que foi projetado para atender a uma demanda pontual, não genérica.

O polimorfismo do tipo ad-hoc é o polimorfismo que ocorre em tempo de compilação.

Técnicas *ad hoc* serão sempre aquelas para resolver a necessidades pontuais.

Esta categoria abrange duas técnicas

- Coerção (também chamado de *casting*)
- Sobrecarga.

Polimorfismo de coerção (coercion)

Essa conversão acontece de maneira implícita, dizemos que existe polimorfismo por coerção já que você está “coergindo” ou convertendo o tipo de uma variável para outra.

```
int a = 2;
double n = a;
```

Isso só é possível porque int é ‘menor’ que double.

O contrário não é possível na linguagem Java!

Upcast

Converter sem perda de informação

```
int i = 20;
double x =2.5;
x = (double) i;
System.out.println(x);
```

Downcast

converter com perca de informação

```
int i = 20;
double x = 2.5;
i = (int) x;
System.out.println(i);
```

Polimorfismo de sobrecarga (Overloading)

Possuímos métodos com o mesmo nome

Assinaturas são diferentes

Esse é o conceito de polimorfismo de sobrecarga!

É executado em tempo de compilação (Antes de executar o código você já sabe qual método será executado).

```
public class Animal {
      public void comer() {
    System.out.println("Comendo...");
  }

  public void comer(String nomeComida) {
    System.out.println("Comendo " + nomeComida);
  }
  public static void main(String args[]){
      Animal a = new Animal();
      a.comer();
      a.comer("Batata");

  }
}
```

Sobrecarga

Note os retornos dos métodos.

Qual a saída?

```
public class Animal {
      public void comer() {
    System.out.println("Comendo...");
  }

  public int comer(String nomeComida) {
    System.out.println("Comendo " + nomeComida);
    return 0;
  }
  public static void main(String args[]){
    Animal a = new Animal();
    System.out.println(a.comer("batata"));
  }
}
```

Ponto2D

Análise o código ao lado.

Torne este código funcional.

Implemente conceitos de polimorfismo e encapsulamento de dados.

Programe a classe com a maior quantidade de polimorfismo e encapsulamento possível.

```
public class Ponto2D {
    private double x;
    private double y;
    Ponto2D(){
        System.out.println("Construtor vazio");
    }
    Ponto2D(double x, double y){
        System.out.println("Construtor double");
    }
    Ponto2D(int x, int y){
        System.out.println("Construtor int");
    }
    Ponto2D(float x, float y){
        System.out.println("Construtor float");

    }
    public double CalcularDistancia(Ponto2D p1, Ponto2D p2){
        return 0;
    }
      public double CalcularDistancia(double x1, double x2,double y1, double y2){
        return 0;
    }
    public static void main(String args[]){
        Ponto2D p = new Ponto2D();
         p = new Ponto2D(1.1,1);
         p = new Ponto2D(1.2f,1);
    }
}
```

Exercícios.

Crie a Classe calculadora, esta classe deve ser capaz de realizar as operações básicas de matemática. Utilize conceitos de encapsulamento, construtores e polimorfismo.