Polimorfismo

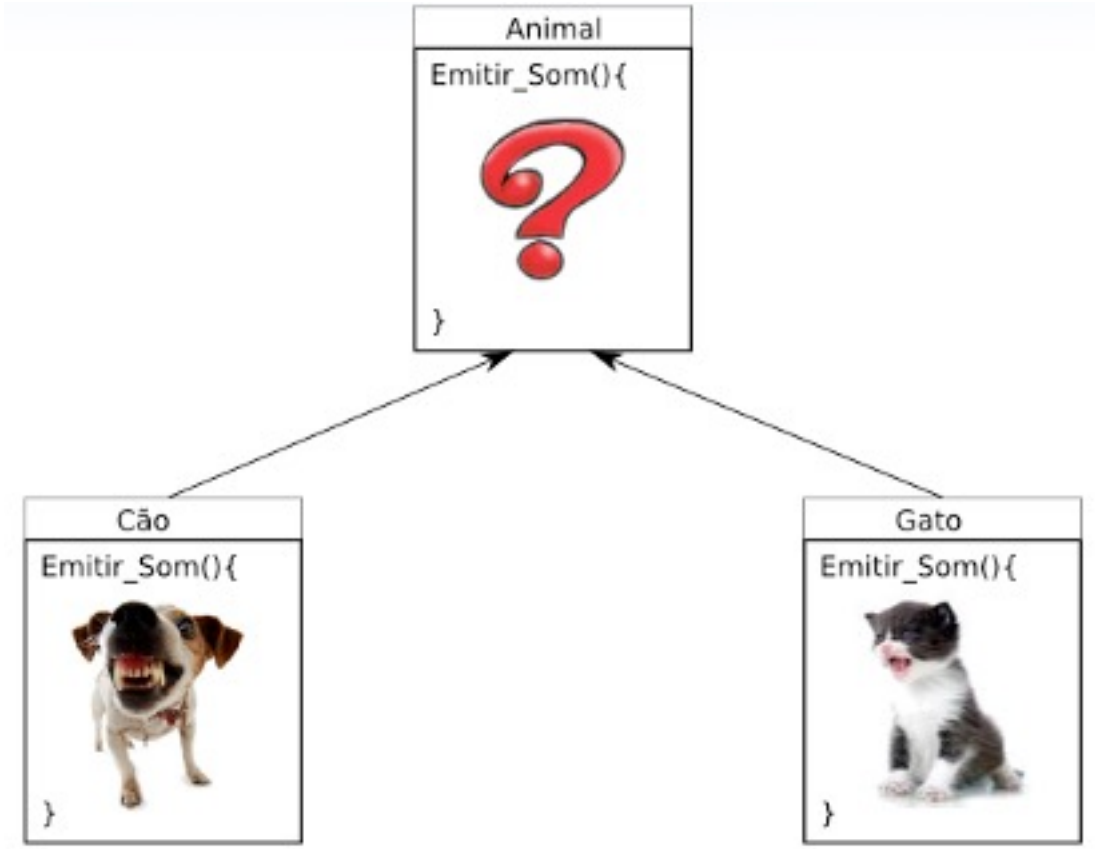
Profa. Karen Selbach Borges

Última Atualização: Abr/2021

Polimorfismo

- Significa oferecer diferentes formas (implementações) para métodos fornecidos pela superclasse.

Exemplo



Polimorfismo – Overriding

- Sobrescrita: uma classe filha pode oferecer uma outra implementação para um método herdado.
- Importante : o método deve ter a mesma assinatura (nome, parâmetros e valor de retorno). Entretanto:
  - O modificador de acesso **pode** ser alterado para menos restritivo. Por exemplo de protected para public.

Exemplo


```
public class Animal {
   protected void eat() {
   System.out.println("Animal comendo");
}

public class Horse extends Animal {
   public void eat() {
       System.out.println(”Cavalo comendo");
   }
}
```

Palavra Reservada “final”

- Métodos declarados como final não aceitam ser sobrescritos.
- Entretanto eles podem ser sobrecarregados.

```
public class Animal {
   public final void eat() {
   System.out.println("Animal comendo");
}

public class Horse extends Animal {
   public void eat() {
      System.out.println(”Cavalo comendo");
   }
}
```

Erro !

Polimorfismo – Overloading

- Sobrecarga: é possível a existência de vários métodos de mesmo nome, porém com assinaturas levemente diferentes. Observe que:
  - A lista de argumentos **deve** ser alterada.
  - O tipo de retorno **pode** ser alterado.
  - O modificador de acesso **pode** ser alterado.
  - Novas exceções podem **ser** lançadas.

Exemplo

```
public class Animal {
   public void eat() {
   System.out.println("Animal comendo");
}

public class Horse extends Animal {
   public void eat(String s) {
     System.out.println(”Cavalo comendo" + s);
   }
}
```

Sobrecarga de Métodos

- É possível que dentro de uma mesma classe exista mais de uma versão para um mesmo método

```
public class Animal {
   public void eat() {  System.out.println("Animal comendo");  }

public class Horse extends Animal {
   public void eat(String s) { System.out.println(”Cavalo comendo" + s); }
   public void eat(String s, double q) { System.out.println(”Cavalo
comendo" + q + “Kg de + s); }
}
```

Importante

Fica a cargo do compilador escolher o método a ser utilizado levando em consideração a hierarquia de classes e as assinaturas dos métodos.

```
public class Animal {
   public void eat () { //…}
}

public class Horse extends Animal {
   public void eat (String s) { //… }
   public void eat (double q) { //… }
}

public class Poney extends Horse {
    public void eat (String s, double q) { //…. }
}
```

Importante

```
public class Principal {
   public static void main (String [ ] args){
      Horse cavalo = new Horse ( );
      cavalo.eat(); // método herdado de Animal
      cavalo.eat(“feno”); // primeiro método eat do Cavalo
      cavalo.eat(10.5); // segundo método eat do Cavalo
      Poney ponei = new Poney ( );
      ponei.eat(); // método herdado de Animal
      ponei.eat(“alfafa”); // método herdado do Cavalo
      ponei.eat (5.3); // método herdado do Cavalo
      ponei.eat(“alfafa”, 5.3); // método do próprio Ponei
   }
}
```

POLIMORFISMO NA PRÁTICA

Voltando à Classe Object

- Alguns métodos:
  - equals : verifica se dois objetos possuem o mesmo OID
    - Sempre vai retornar FALSE
  - toString : retorna o objectID
    - O OID é uma informação de pouca utilidade

Sobrescrita de métodos

- É uma boa prática de programação sobrescrever o método *equals*, dentro das classes que criamos, comparando o conteúdo dos objetos.
- Da mesma forma, deve-se sobrescrever o método *toString*, retornando uma String contendo os valores das variáveis de instância dos objetos.

Exemplo de toString

```
public class Cliente {
    private String nome;
    private String cpf;

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

    public String toString(){
         return ("Nome : "+ this.nome +
                    "\nCpf:" + this.cpf);
    }

}
```

Serve para concatenar as Strings. Se a variável for numérica, ela será automaticamente convertida em String para permitir a concatenação.

Exemplo de equals

```
public class Cliente {
    private String nome;
    private String cpf;

    public boolean equals(Object obj){
        if (obj instanceof Cliente) {              // Testa se dentro de obj existe um cliente
            Cliente cli = (Cliente) obj;           // Converte obj para Cliente (cast)
            return (this.cpf.equals(cli.cpf));     // compara as Strings de cpf.
        }
        else return false;
    }
}
```

- Objetos são sempre comparados usando equals.
- Tipos primitivos (int, double, boolean, etc) devem ser comparados usando ==.