Introdução a Programação Orientada a Objetos

Prof. Joice Seleme Mota

Bem vindos a Objetolândia
Princípios Básicos da Orientação a Objetos

- Herança
- Polimorfismo
- Encapsulamento

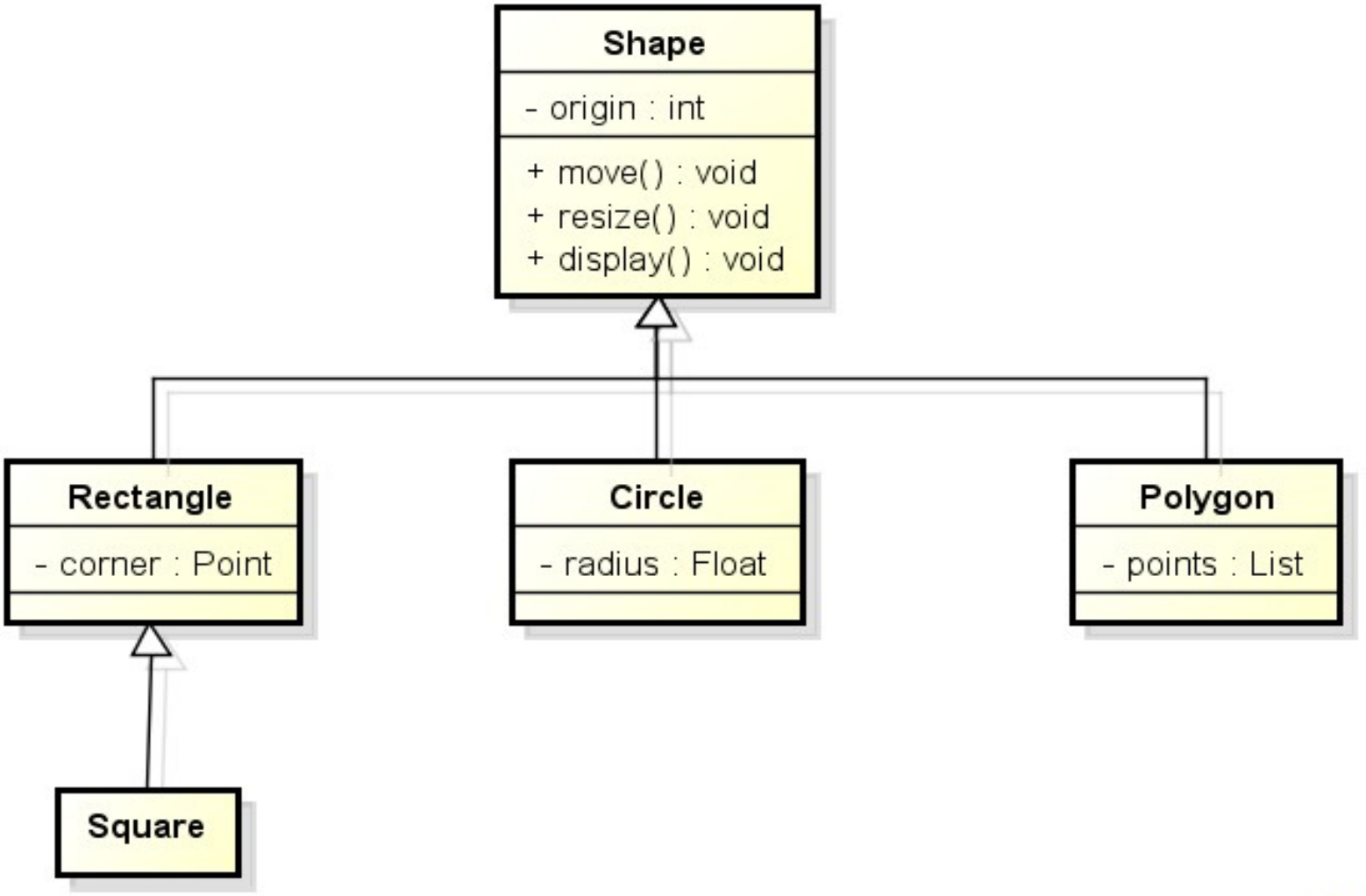
Bem vindos a Objetolândia
Herança

- Quando uma classe herda o comportamento de outra

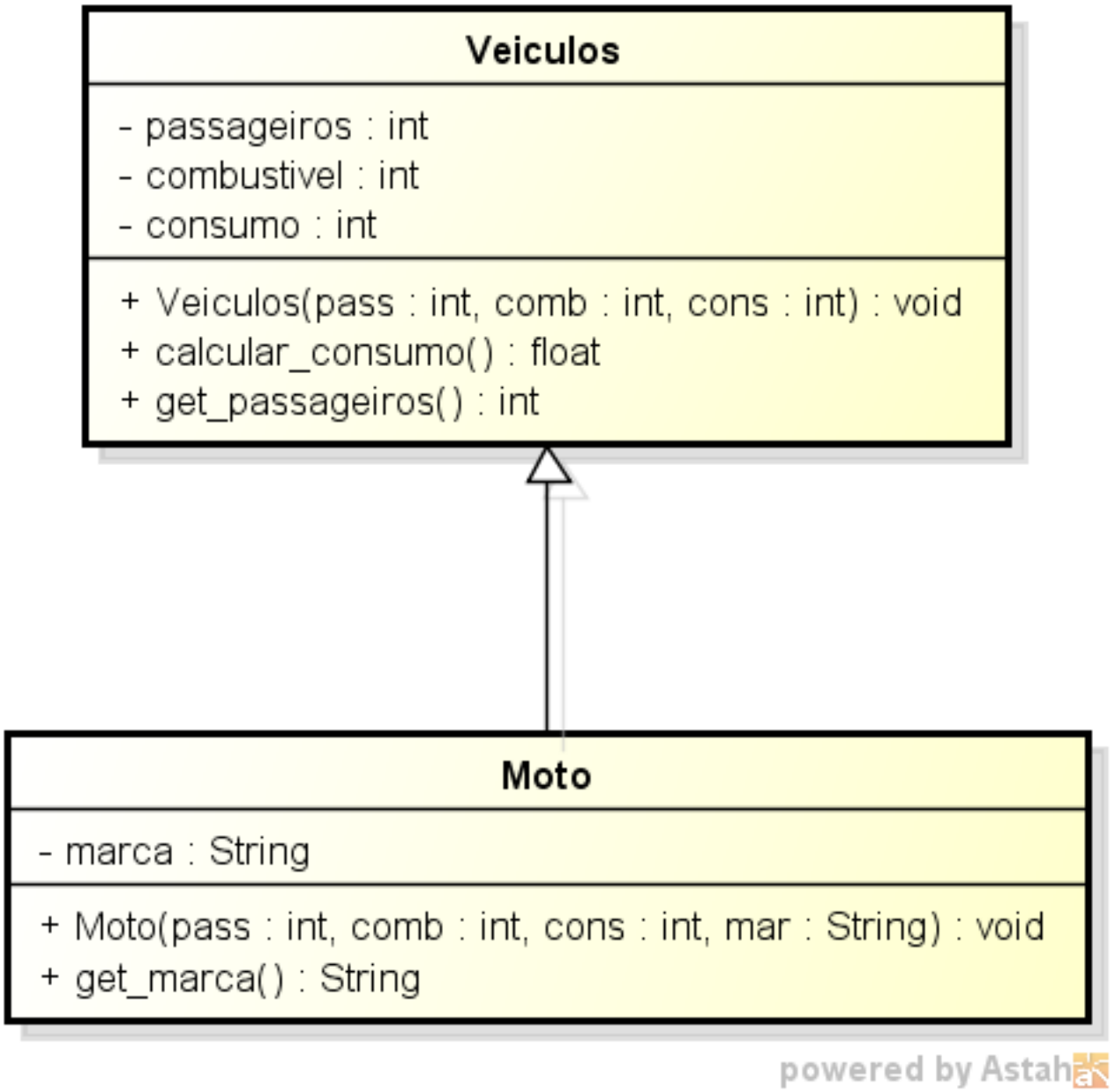
- Pode alterar o comportamento se necessário

- Permite que se crie classes baseadas em outras classes, evitando a duplicação e repetição de código

Bem vindos a Objetolândia
Herança



Implementando Herança Simples



Implementando Herança Simples

A classe veículos

```
public class Veiculos {

        private int passageiros;
        private int combustivel;
        private int consumo;

   public Veiculos(int pass, int comb, int cons){
       this.passageiros = pass;
       this.combustivel = comb;
       this.consumo = cons;
   }

   public int calcular_consumo(){
       return (this.combustivel*this.consumo);
   }

  public int get_passageiros(){
      return this.passageiros;
  }
}
```

Implementando Herança Simples

A classe Moto

```
public class Moto extends Veiculos {
    private String marca;

    public Moto(int pass, int comb, int cons, String mar) {
        super(pass, comb, cons);
        this.marca=mar;
    }

    public String get_marca(){
         return this.marca;
     }
}
```

Implementando Herança Simples

Testando

```
public static void main(String[] args) {

    Veiculos minivan = new Veiculos(7,80,9);
    Veiculos sportcar = new Veiculos(2,9,35);
    Moto moto = new Moto (2,10,20,"honda");
    System.out.print("Minivan pode transportar " + minivan.get_passageiros() + "
passageiros ");
    System.out.println("com uma autonomia de " + minivan.calcular_consumo() + "
quilômetros");
    System.out.print("Carro esporte pode transportar " + sportcar.get_passageiros() +
" passageiros ");
    System.out.println("com uma autonomia de " + sportcar.calcular_consumo() + "
quilômetros");
    System.out.print("Moto " + moto.get_marca() + " pode transportar " +
moto.get_passageiros() + " passageiros ");
    System.out.println("com uma autonomia de " + moto.calcular_consumo() + "
quilômetros");
    }
}
```

Minivan pode transportar 7 passageiros com uma autonomia de 720 quilômetros
Carro esporte pode transportar 2 passageiros com uma autonomia de 315 quilômetros
Moto honda pode transportar 2 passageiros com uma autonomia de 200 quilômetros

Bem vindos a Objetolândia
Polimorfismo

- Enviar uma mesma mensagem para diferentes objetos e fazê-los responder da maneira correta.
- Quando uma mesma mensagem pode ser processada de diferentes formas

Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse

Bem vindos a Objetolândia
Polimorfismo

- Usando polimorfismo podemos:
  - Invocar métodos da classe derivada através da classe base em tempo de execução;
  - Permitir que classes forneçam diferentes implementações de métodos que são chamados com o mesmo nome;

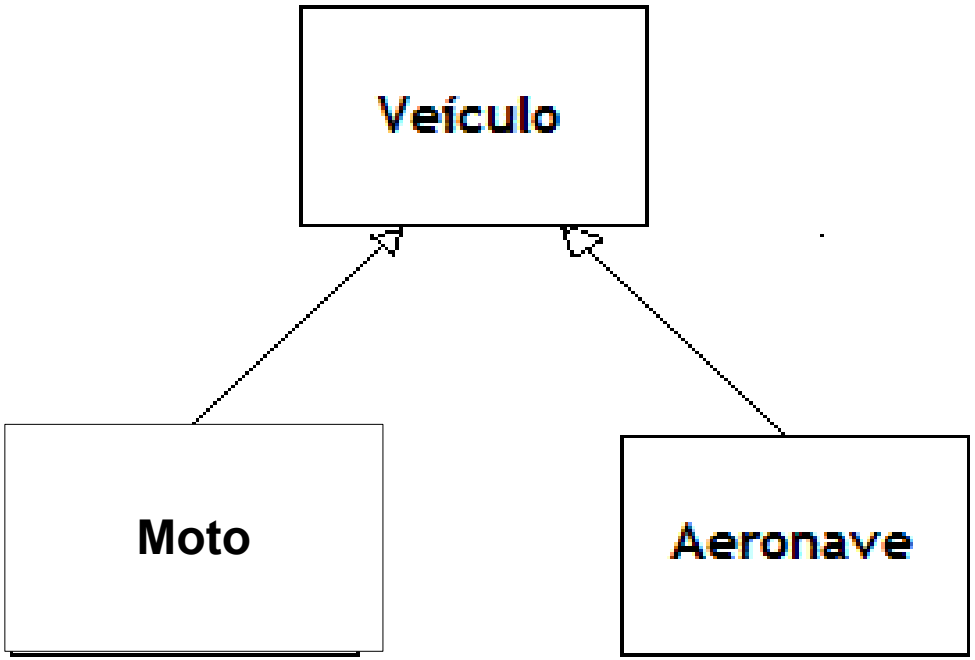
- Existem dois tipos básicos de polimorfismo:
  - Polimorfismo em tempo de compilação (Overloading/Sobrecarga);
  - Polimorfismo em tempo de execução (Overriding/Sobrescrita);

Bem vindos a Objetolândia
Polimorfismo

- O polimorfismo em tempo de compilação
  - utiliza a sobrecarga de métodos e operadores sendo também chamado de ligação precoce (*early binding*).
  - A utilização da sobrecarga de métodos realiza a tarefa com distintos parâmetros de entrada.

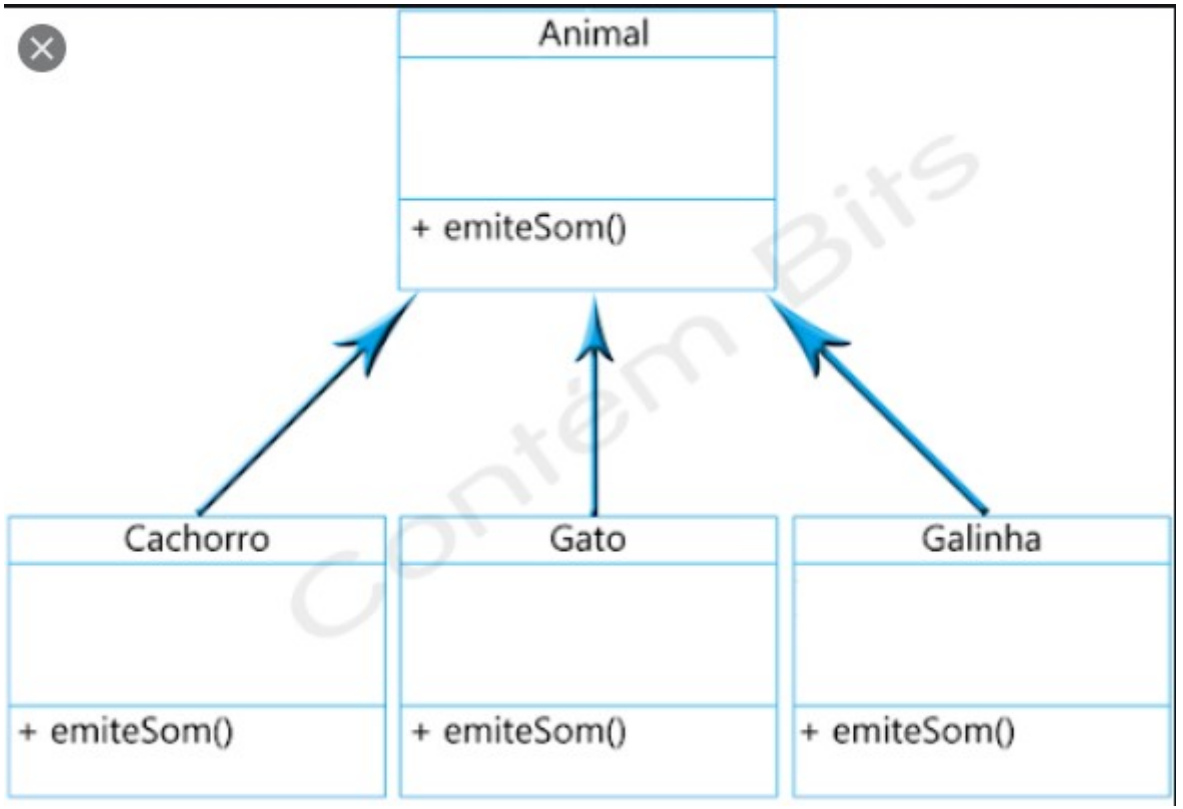
- O polimorfismo em tempo de execução
  - pode ser feito usando herança e métodos virtuais.
  - Quando sobrescrevemos(override) os métodos virtuais estamos alterando o comportamento dos métodos para a classe derivada. Isto também é conhecido como ligação tardia (*late binding*).

Bem vindos a Objetolândia
Polimorfismo



Se tivéssemos o método Mover() em todas as classes, Mover() tem a mesma assinatura, porém os objetos se movem de maneira diferente. (Sobreposição)

Bem vindos a Objetolândia
Polimorfismo



Todos os animais emitem som. O comportamento de emitir som é o mesmo, mas a forma como emitem som é diferente, por isso cada um tem o seu som

Bem vindos a Objetolândia
Polimorfismo

```
public float calcular_preco(float valor, int imposto){
    return valor + imposto;
 }

 public float calcular_preco(float valor, int imposto, int seguro){
    return valor + imposto+seguro;
 }
```

O método Calcular_preço () tem por objetivo mostrar o preço final do veiculo mas aceita parâmetros de entrada diferentes (Sobrecarga)

Bem vindos a Objetolândia
Encapsulamento

- Encapsular significa "ocultar informações"

- O encapsulamento acontece quando se oculta parte dos dados do resto da aplicação

- Limita a capacidade de outras partes do código acessarem esses dados

O encapsulamento separa seus dados
do comportamento da
sua aplicação

Bem vindos a Objetolândia
Encapsulamento

- Consiste em ocultar a implementação dos métodos e restringir o acesso aos atributos somente através de métodos internos

- Para isso precisamos que todos os atributos sejam privados à classe;

1) Classe Pessoa

```
public class Pessoa{
   private String nome;
   public String getNome(){
      return this.nome;
   }
   public void setNome(String nome){
      this.nome = nome;
   }
}
```

2) Classe Principal

```
public class Principal{
   public static void main(String[] args){
         Pessoa p = new Pessoa();
         p.setNome("Joaozinho");
         System.out.println("Nome :"  + p.getNome() );
   }
}
```