Orientação a Objetos

Revisão ...

Programação Orientada a Objetos é um paradigma de programação centrado em torno dos conceitos de classes e objetos

Objetos são unidades de programa que abstraem dados e operações relacionados

Objetos possuem características e comportamentos

Objetos possuem características e comportamentos

Ex. Bicicleta:

CARACTERÍSTICAS – atributos velocidade e marcha;

COMPORTAMENTO – exposto pelos métodos: mudarMarcha, aumentarVelocidade.

Uma classe é uma abstração de uma coleção de objetos com características e comportamentos em comum

As características de uma classe se manifestam computacionalmente na forma de atributos

O comportamento da classe é exposto através de métodos (funções)

Uma classe fornece um esquema para construção de objetos com características e comportamentos semelhantes

Um objeto é uma instância de uma única classe

Um programa OO é um conjunto de objetos que colaboram entre si para a solução de um problema

Objetos colaboram através de trocas de mensagens

A troca de mensagem é realizada através da chamada de um método

Encapsulamento:
Princípio pelo qual cada componente de um programa deve agregar toda a informação relevante para sua manipulação.

Ocultação da Informação:
princípio pelo qual cada componente deve manter oculta sob sua guarda uma decisão de projeto única. Para a utilização desse componente, apenas o mínimo necessário para sua operação deve ser revelado (tornado público).

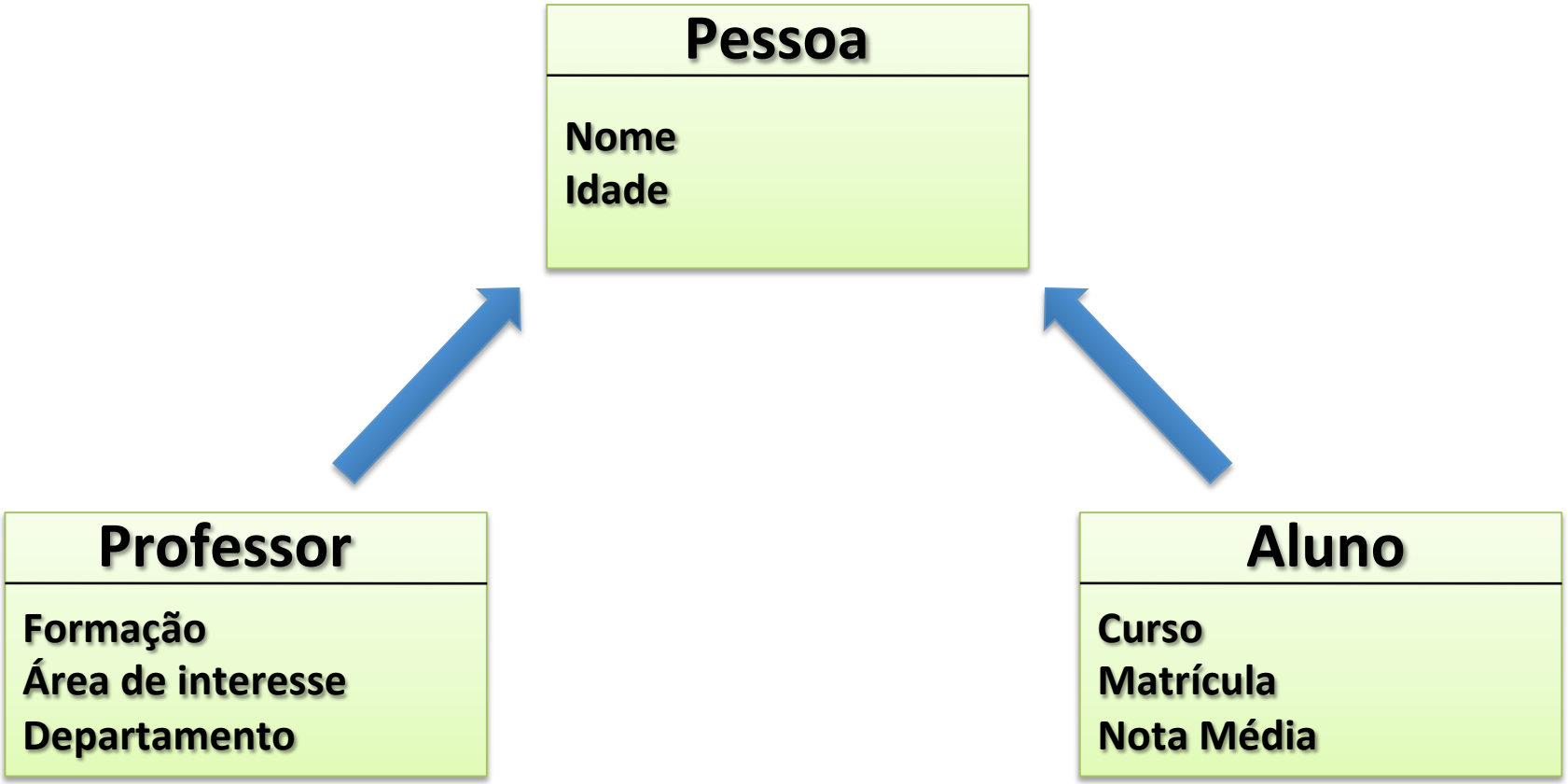

Identidade
Classificação
Encapsulamento
Polimorfismo
Hereditariedade
Ligação Dinâmica

Diferentes classes podem ter diversas semelhança

Duas ou mais classes poderão compartilhar os mesmos atributos e/ou os mesmos métodos.

Herança: Permite a uma classe herdar o estado (atributos) e o comportamento (métodos) de outra classe.

- Superclasse
- Subclasse
- Ancestral
- Descendente

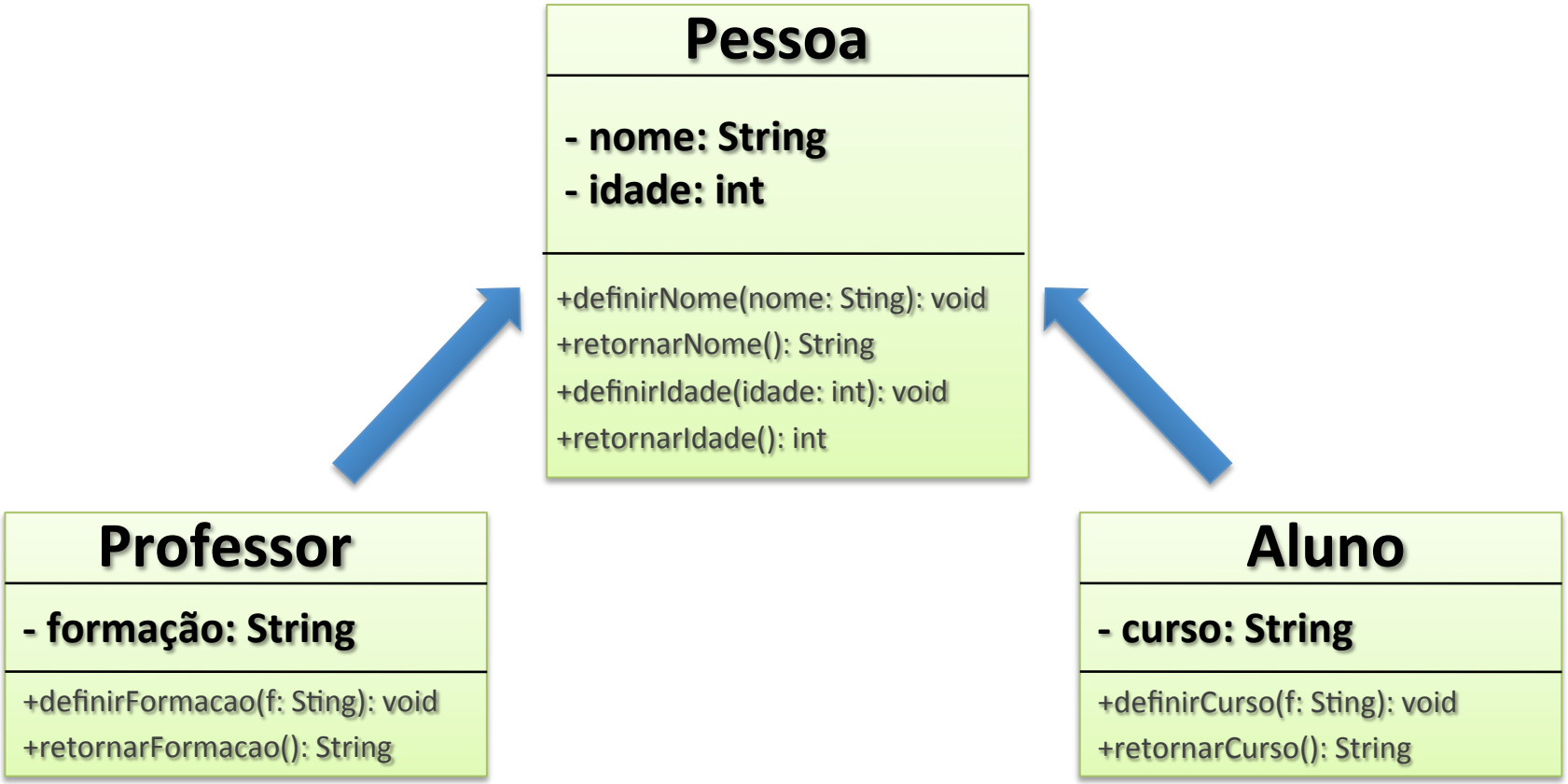

Pessoa
Nome
Idade
Professor
Formação
Área de interesse
Departamento
Aluno
Curso
Matrícula
Nota Média

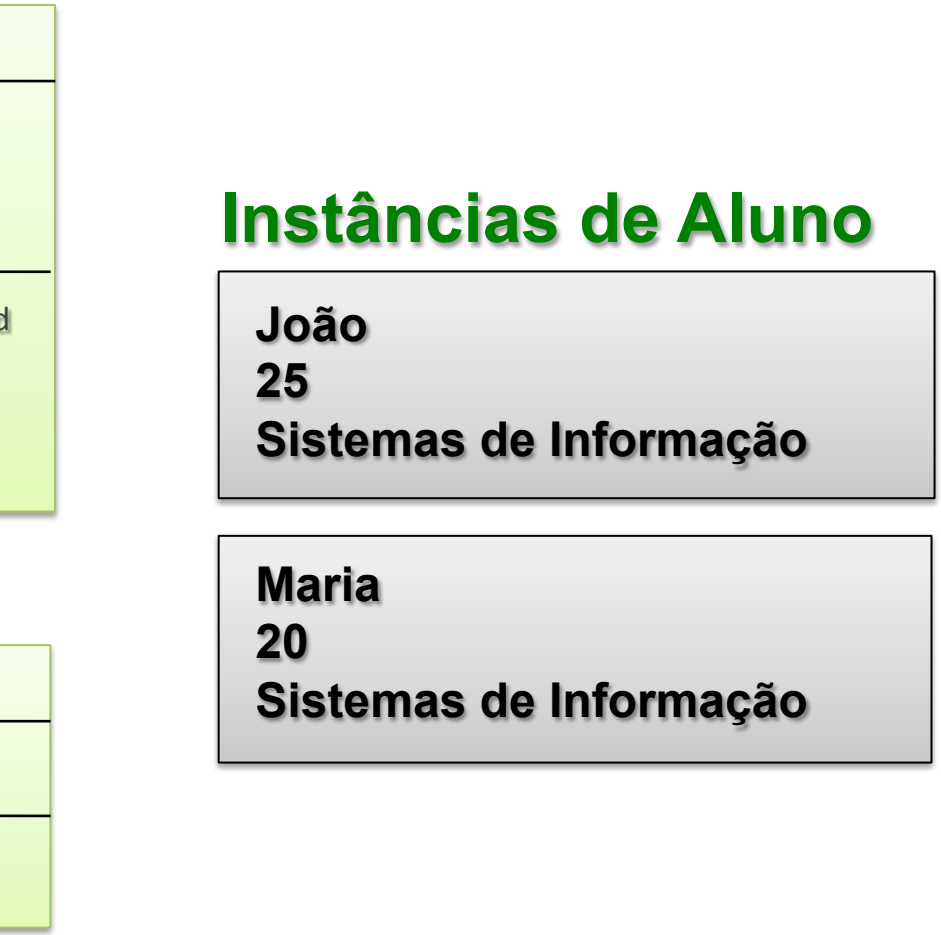
Professor
- nome: String - idade: int - formação: String
+definirNome(nome: Sting): void +retornarNome(): String +definirIdade(idade: int): void +retornarIdade(): int +definirFormacao(f: Sting): void +retornarFormacao(): String ...

Aluno
- nome: String - idade: int - curso: String
+definirNome(nome: Sting): void +retornarNome(): String +definirIdade(idade: int): void +retornarIdade(): int +definirCurso(c: Sting): void +retornarCurso(): String ...

Professor
- nome: String - idade: int - formação: String
+definirNome(nome: Sting): void +retornarNome(): String +definirIdade(idade: int): void +retornarIdade(): int +definirFormacao(f: Sting): void +retornarFormacao(): String ...

Aluno
- nome: String - idade: int - curso: String
+definirNome(nome: Sting): void +retornarNome(): String +definirIdade(idade: int): void +retornarIdade(): int +definirCurso(c: Sting): void +retornarCurso(): String ...


Pessoa
- nome: String
- idade: int
+definirNome(nome: Sting): void
+retornarNome(): String
+definirIdade(idade: int): void
+retornarIdade(): int
Professor
- formação: String
+definirFormacao(f: Sting): void
+retornarFormacao(): String
Aluno
- curso: String
+definirCurso(f: Sting): void
+retornarCurso(): String



Instâncias de Aluno

João
25
Sistemas de Informação

Maria
20
Sistemas de Informação

```
//SuperClass.java
public class SuperClass
{
   // Atributos e métodos
}

//SubClass.java
public class SubClass extends SuperClass
{
   // Atributos e métodos
}
```

Orientação a Objetos

Herança

```
class Pessoa {
    String  nome;
    int    idade;

    void definirNome(String valor) {
             nome = valor;
      }

    String retornarNome() {
             return nome;
      }

    void definirIdade(int valor) {
             idade = valor;
      }

    int retornarIdade() {
             return idade;
      }
}
```

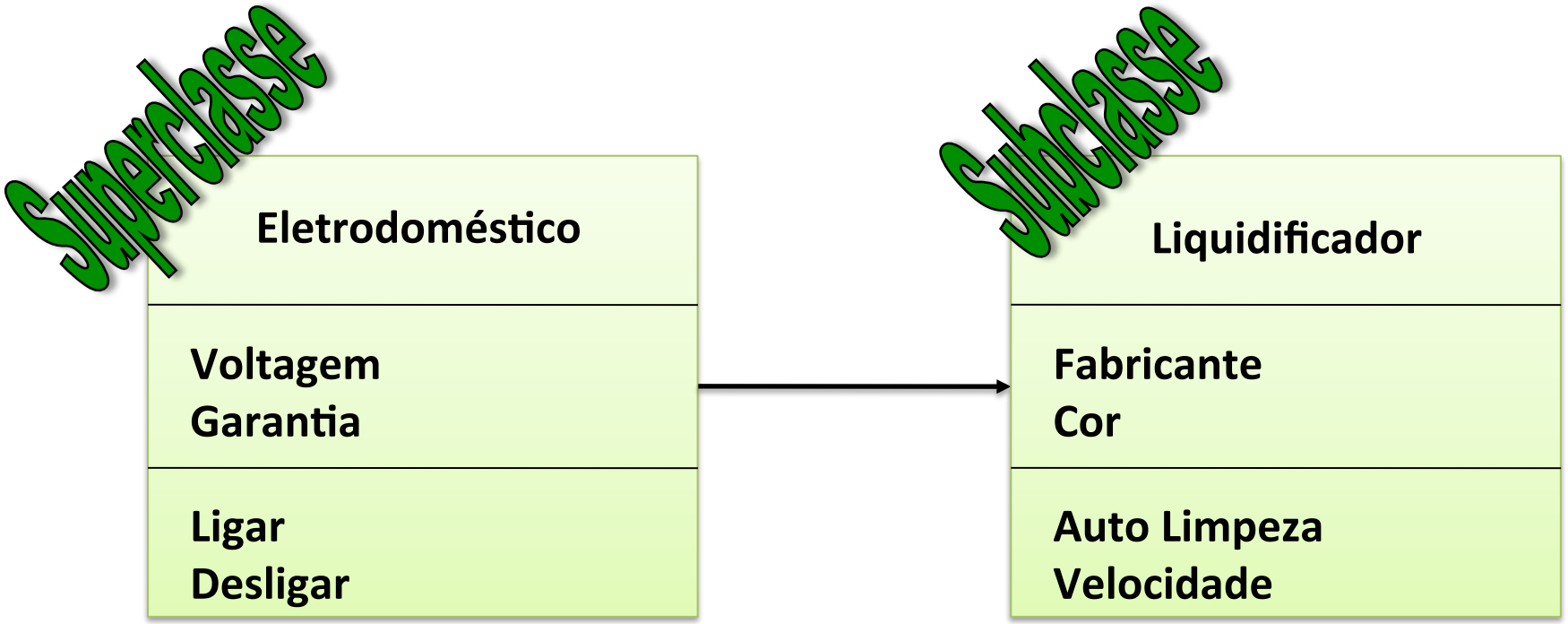
```
class Aluno extends Pessoa {
    String   curso;

    void definirCurso(String valor) {
             curso = valor;
      }

    String retornarCurso() {
             return curso;
      }

}
```

```
Aluno joao = new Aluno();

joao.definirNome("João");
joao.definirIdade(25);
joao.definirCurso("Sistemas de
Informação");
```

João
25
Sistemas de Informação

```
Aluno maria = new Aluno();

maria.definirNome("Maria");
maria.definirIdade(20);
maria.definirCurso("Ciência da
Computação");
```

Maria
20
Ciência da Computação


Superclasse
Eletrodoméstico
Voltagem
Garantia
Ligar
Desligar
Subclasse
Liquidificador
Fabricante
Cor
Auto Limpeza
Velocidade

Superposição de atributos e métodos

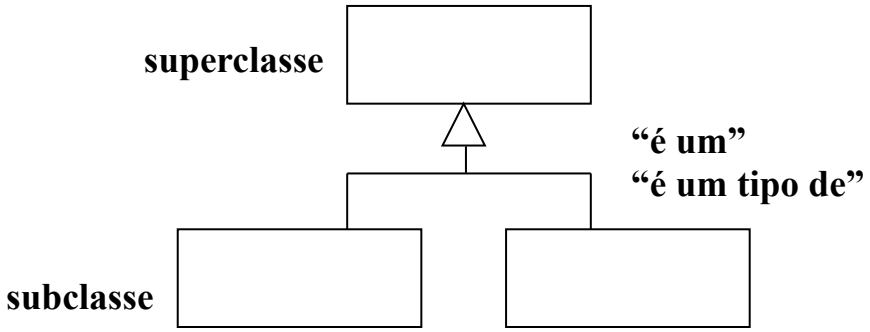
Atributos com mesmo nome de atributos de superclasses escondem o atributo da superclasse

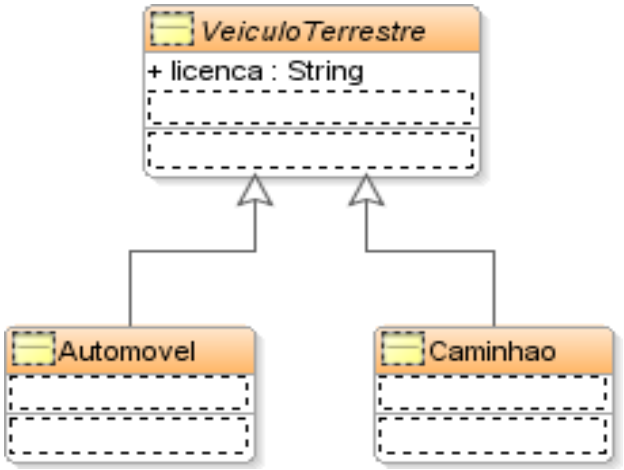
O acesso ao atributo da superclasse se faz com o uso do operador **super**

```
public class Automovel extends VeiculoTerrestre {

    public String licenca;

    public Automovel(){
      super.licenca="LAL1229";
    }
}
```

Uma generalização é um relacionamento entre itens gerais (superclasse) e itens mais específicos (subclasses)

É representada por uma linha sólida com um triângulo vazado apontando para o item mais geral.





São classes que existem apenas conceitualmente para definir conjuntos de métodos e propriedades

Uma *classe abstrata* é uma classe que *não tem instâncias diretas*

Entretanto, suas classes descendentes podem ter instâncias diretas

```
public abstract class Empregado {
    public Empregado (String nome, double salario) {
    this.numero = ++contador;
     this.nome = nome;
     this.salario = salario; }

     public abstract void aumentaSalario(double percentual);
     }

     public class Vendedor extends Empregado{
       public void aumentaSalario (double percentualDeAumento)
       { percComissao = percComissao * (1+percentualDeAumento/100);}
     }

     public class Gerente extends Empregado {
       public void aumentaSalario(double percentual) {
         double novoSalario = getSalario() * (1+2 * percentual/100);
           setSalario(novoSalario);
     }
}
```

Um interface *apenas propõe os métodos que devem ser implementados* pelas classes que desejarem

Uma interface define um tipo.

```
public interface Ordenavel {
    public int comparar(Ordenavel b);
}

public class Empregado implements Ordenavel {
    ...
    public int comparar(Ordenavel b) {
         Empregado e = (Empregado)b;
         if (salario < e.salario) return -1;
         if (salario > e.salario) return 1;
         return 0;
    } ...
}

public class Ordenacao {
    public static void ordena(Ordenavel[] a)
    {  ... if (a[i].comparar(a[i+]) > 1) ... }
}
```

Classes **Abstratas** Vs Classes **Concretas**

- Uma *classe abstrata* é uma classe que *não tem instâncias diretas*, mas cujas classes descendentes podem ter instâncias diretas.
- Uma *classe concreta* é uma *classe que pode ser instanciada*.

Classes **Abstratas** Vs **Interfaces**

- A *classe abstrata* pode possuir métodos não abstratos, bastando ter apenas um método abstrato para ser considerada como tal.
- Um interface *apenas propõe os métodos que devem ser implementados* pelas classes que desejarem.
- Uma interface define um tipo.

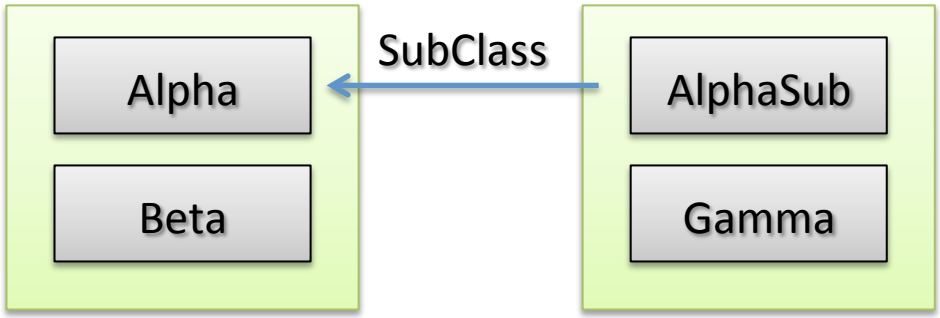
Uma classe pode ser:

- ✓ **public**
  acessada por qualquer outra classe.
- ✓ **nenhum modificador** (package-private)
  acessada somente dentro do seu pacote.

Atributos e Métodos podem ser:

- ✓ **public**
  acessados por qualquer outra classe.
- ✓ **nenhum modificador** (package-private)
  acessados somente dentro do seu pacote.
- ✓ **private**
  acessados somente dentro de suas próprias classes.
- ✓ **protected**
  acessados somente dentro do seus pacotes e por suas subclasses.

modificador	Classe/ Atributos ou métodos	pacote	subclasse	todos
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	Não
nenhum	Sim	Sim	Não	Não
private	Sim	Não	Não	Não



modificador	Alpha	Beta	AlphaSub	Gamma
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	Não
nenhum	Sim	Sim	Não	Não
private	Sim	Não	Não	Não

```
class Pessoa {
   String nome;
   int     idade;

   public Pessoa (String nome, int idade)
   {
      this.nome = nome;
      this.idade = idade;
   }

   public Pessoa ()
   {
      this.nome = "João";
      this.idade = 25;
   }

}
```

```
Pessoa maria =
   new Pessoa("Maria", 20);

Pessoa joao = new Pessoa();
```

Devem ter o mesmo nome da classe que inicializam.

Podem ter parâmetros.

Não tem retorno.

Se não é declarado nenhum construtor, a linguagem provê um construtor padrão sem argumentos que não faz nada.

Orientação a Objetos

Construtores

Pacotes

Classe

Variáveis

Construtores

Métodos

```
import java.util.*;

public class AloMundo {

   private String mensagem = “ ”;

   public AloMundo () {
          Date data = new Date();
          mensagem =  "Alô, Mundo” + data.toString() ;
   }

   public void mostrarMensagem ()  {
          System.out.println( mensagem );
     }
}
```

Orientação a Objetos

Construtores

```
public class Pessoa {
    private  String nome;
    private  int idade;

    void definirNome(String valor) {
            nome = valor;
     }

    String retornarNome() {
            return nome;
     }

    void definirIdade(int valor) {
            idade = valor;
     }

    int retornarIdade() {
            return idade;
     }
```

```
 public static void main (String[] args) {
    Pessoa p1 = new Pessoa();
     p1.definirNome("João");
     p1.definirIdade(25);
     System.out.println( p1.retornarNome() +
     "  " + p1.retornarIdade());

      Pessoa p2 = new Pessoa();
       p2.definirNome("Maria");
       p2.definirIdade(20);
       System.out.println(p2.retornarNome() +
        "  " + p2.retornarIdade());
  }

} // fim da classe Pessoa
```