

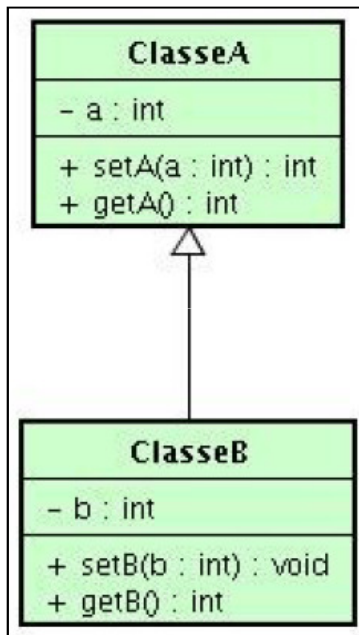
Herança e Polimorfismo



Prof. Bruno Gomes
bruno.gomes@ifrn.edu.br

Programação Orientada a Objetos

Revisando



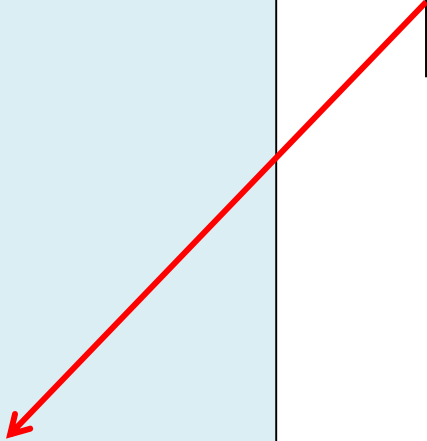
**extends é usado para
indicar herança em JAVA**

```
class ClasseA {
    protected int a;
    public int getA() {
        return a;
    }
    public void setA(int a) {
        this.a = a;
    }
}
class ClasseB extends ClasseA{
    private int b;
    public int getB() {
        return b;
    }
    public void setB(int b) {
        this.b = b;
    }
}
```

Revisando

```
public static void main(String args[]) {  
    ClasseA a = new ClasseA();  
    ClasseB b = new ClasseB();  
    a.setA(10);  
    b.setA(20);  
    b.setB(30);  
    System.out.println(a.getA());  
    System.out.println(b.getB());  
    System.out.println(b.getA());  
}
```

Método herdado de
ClasseA



Todos os membros definidos em ClasseA
também existem em ClasseB

Revisando

- Uma classe que herda de outra é um subtipo
- Podemos ter uma variável do tipo ClasseA que referencia um objeto do tipo ClasseB

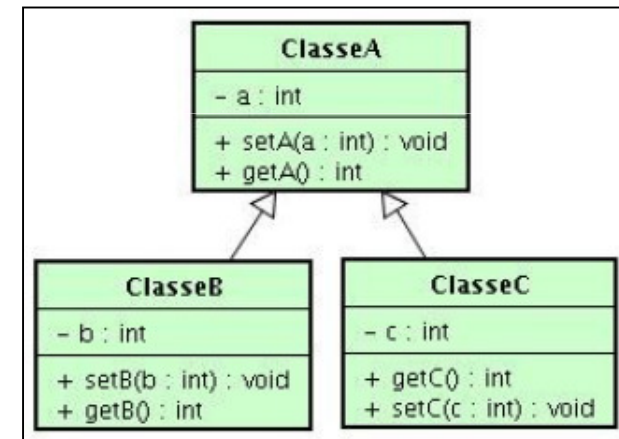
variável b NÃO
executa métodos
de ClasseB, pois
ela só conhece a
interface de
ClasseA

```
ClasseA a,b;  
a = new ClasseA();  
b = new ClasseB();  
a.setA(10);  
b.setA(20);  
//b.setB(30);  
System.out.println(a.getA()+b.getA());
```

Revisando

- Operador instanceof
 - Determina se um objeto obj é de uma classe Cla

```
public static void main(String args[]){  
    ClasseA obj;  
    obj = new ClasseC();  
    if (obj instanceof ClasseA)  
        System.out.println("obj é ClasseA");  
    if (obj instanceof ClasseB)  
        System.out.println("obj é ClasseB");  
    if (obj instanceof ClasseC)  
        System.out.println("obj é ClasseC");  
}
```



obj é da *ClasseA*
e da *ClasseC*

Revisando

- Toda classe JAVA tem pelo menos um construtor
- Todo construtor **deve chamar o construtor da superclasse**
 - Deve ser o primeiro comando do construtor
 - Compilador coloca código caso o programador não coloque



```
class ClasseD extends ClasseA{
    public ClasseD(){
        super();
        /* ... */
    }
}
```


```
class ClasseD extends ClasseA{
    public ClasseD(){
        /* ... */
    }
}
```

Revisando

- Acessando Construtores da superclasse:
 - Utilizar **super()**;

```
class ClasseA {  
    private int a;  
  
    public ClasseA(int a){  
        this.a=a;  
    }  
  
    public int getA() {  
        return a;  
    }  
    public void setA(int a) {  
        this.a = a;  
    }  
}
```

```
class ClasseB extends ClasseA{  
    private int b;  
  
    public ClasseB(int a, int b){  
        super(a);  
        this.b=b;  
    }  
  
    public int getB() {  
        return b;  
    }  
    public void setB(int b) {  
        this.b = b;  
    }  
}
```



Sobrescrita

- Podemos redefinir um método na subclasse
 - Sobrescrita do método
 - Muda comportamento definido na superclasse

```
class ClasseB extends ClasseA{  
    private int b;  
    public int getB() {  
        return b;  
    }  
    public void setB(int b) {  
        this.b = b;  
    }  
    public void setA(int a) {  
        this.a = a+b;  
    }  
}
```

```
ClasseB b = new ClasseB();  
b.setB(10);  
b.setA(10);  
System.out.println(b.getA());
```

Valor impresso será 20

Sobrescrita

- É possível chamar o método da superclasse
 - Palavra **super**
 - `super.metodo();`
- Pode ser usado em qualquer método
 - Podemos sobrescrever o método para adicionar funcionalidade, mantendo a funcionalidade existente

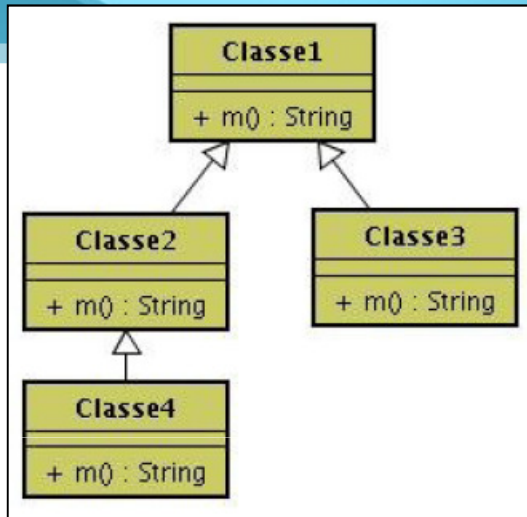
```
class Superclasse{
    /* ...*/
    public void metodo(int x){
        /* ... */
    }
    /* ... */
}
```

```
class Subclasse extends Superclasse{
    public void metodo(int x){
        /* ... */
        super.metodo(10);
        /* ... */
    }
}
```

Polimorfismo

- Muda comportamento
 - Método executado depende da **classe do objeto**
 - Mesma chamada executa métodos diferentes
 - `obj.metodo()` vai executar método que foi definido para **classe do objeto referenciado por obj**
 - Permite executar métodos de subclasses mesmo sem conhecê-las
 - Usado junto com sobrescrita

Polimorfismo



```
Classe1 a, b, c, d;
a = new Classe1();
b = new Classe2();
c = new Classe3();
d = new Classe4();
System.out.println(a.m());
System.out.println(b.m());
System.out.println(c.m());
System.out.println(d.m());
```

```
class Classe1 {
    public String m() {
        return "Classe1";
    }
}
class Classe2 extends Classe1 {
    public String m() {
        return "Classe2";
    }
}
class Classe3 extends Classe1 {
    public String m() {
        return "Classe3";
    }
}
class Classe4 extends Classe2 {
    public String m() {
        return "Classe4";
    }
}
```

Classes Abstratas

- Não podem ser instanciadas
 - Usadas com herança
 - Definir superclasse com características e comportamento comuns e cada subclasse implementa suas especificidades
 - Não existem animais da classe mamífero, apenas de suas subclasses
 - Palavra `abstract` antes do nome da classe


```
public abstract class Mamifero{  
    /* ... */  
}
```

Métodos Abstratos

- Métodos podem ser abstratos
 - Não possuem implementação
 - Devem ser implementados pelas subclasses
 - Ou estas também serão abstratas
 - Se uma classe possuir pelo menos um método abstrato, ela deve ser abstrata
 - Classes abstratas podem ter métodos implementados

```
public abstract class Mamifero{  
    public abstract String metodo();  
}
```

Não possui
implementação



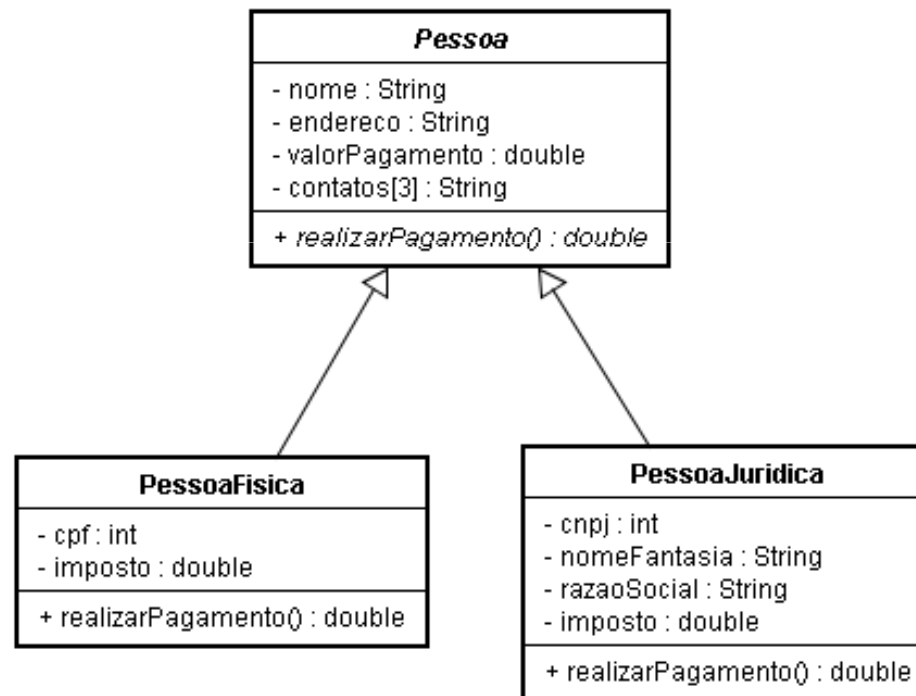
Classes finais

- Não podem ser herdadas
 - Não possuem subclasses
- Não podem ser abstratas
 - Por consequência, não possuem métodos abstratos
- Palavra final antes de class

```
public final class Gato{  
    /* ... */  
}
```

Exercício em Sala 2

- Implementar as classes do seguinte diagrama:



- Detalhes no próximo slide

Exercício em Sala 2

- Detalhes:
 - Classe **Pessoa** e seu método **realizarPagamento** são abstratos;
 - Atributo **contatos[]** é do tipo **String** e corresponde a um *Array* de 3 posições;
 - Atributo **imposto** corresponde a uma porcentagem que será diminuída sobre o **valorPagamento** (10% para PF e 20% para PJ – Estes valores devem ser inicializados no construtor da classe);
 - O método **realizarPagamento** retorna o valor do pagamento diminuído da porcentagem do **imposto**;