INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

Classe Abstrata

Repetindo mais código?

- Como **esta** nossa classe **Funcionario** e **ControleBonficação**;

```
class Funcionario {

    protected String nome;
    protected String cpf;
    protected double salario;

    public double getBonificacao() {
        return this.salario * 1.2;
    }

    // outros métodos aqui

}
```

```
class ControleDeBonificacoes {

   private double totalDeBonificacoes = 0;

      public void registra(Funcionario f) {
         System.out.println("Adicionando bonificacao do funcionario: " + f);
         this.totalDeBonificacoes += f.getBonificacao();
      }

      public double getTotalDeBonificacoes() {
         return this.totalDeBonificacoes;
      }
```

- O método **registra recebe** qualquer **referência** do tipo **Funcionario**, isto é, podem ser **objetos** do tipo **Funcionario** e qualquer de seus **subtipos**: **Gerente**, **Diretor**;

Repetindo mais código?

- Se **não** fosse a **classe Funcionario** precisaríamos criar um **método registra** para **receber** cada um dos tipos de **Funcionario**, um para **Gerente**, um para **Diretor**, etc.

- Utilizamos a classe **Fucionario** apenas nesse **intuito**: de **economizar** um **pouco código** e ganhar **polimorfismo** para criar **métodos** mais **genéricos**, que se **encaixem** a diversos **objetos**.

- Faz **sentido** ter um **objeto** do tipo **Funcionario**?

Repetindo mais código?

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- **Referenciando Funcionario** temos o **polimorfismo** de **referência**. Porém, dar **new** em **Funcionario** pode não fazer sentido;

```
ControleDeBonificacoes cdb = new ControleDeBonificacoes();
Funcionario f = new Funcionario();
cdb.adiciona(f); // faz sentido?
```

- Imagine a classe **Pessoa** e duas **filhas**, **PessoaFisica** e **PessoaJuridica**. A classe **Pessoa**, nesse caso, estaria sendo usada **apenas** para ganhar o **polimorfismo** e **herdar** algumas coisas: não faz **sentido** permitir **instanciá-la.**

- Para resolver esses **problemas**, temos as **classes abstratas**.

Classe abstrata

- O que, **exatamente**, vem a ser a nossa classe **Funcionario**? Nossa **empresa** tem apenas **Diretores**, **Gerentes**, **Secretárias**, etc. Ela é uma **classe** que apenas **idealiza** um **tipo**, define apenas um **rascunho**.

- Para o nosso **sistema**, é **inadmissível** que um objeto seja **apenas** do tipo **Funcionario**.

- Usamos a palavra chave **abstract** para **impedir** que ela **possa** ser **instanciada**.

Classe abstrata

```
abstract class Funcionario {

    protected double salario;

    public double getBonificacao() {
        return this.salario * 1.2;
    }

    // outros atributos e metodos comuns a todos Funcionarios

}
```

E, no meio de um código:

```
Funcionario f = new Funcionario(); // não compila!!!
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot instantiate the type Funcionario

        at br.com.caelum.empresa.TestaFuncionario.main(TestaFuncionario.java:5)
```

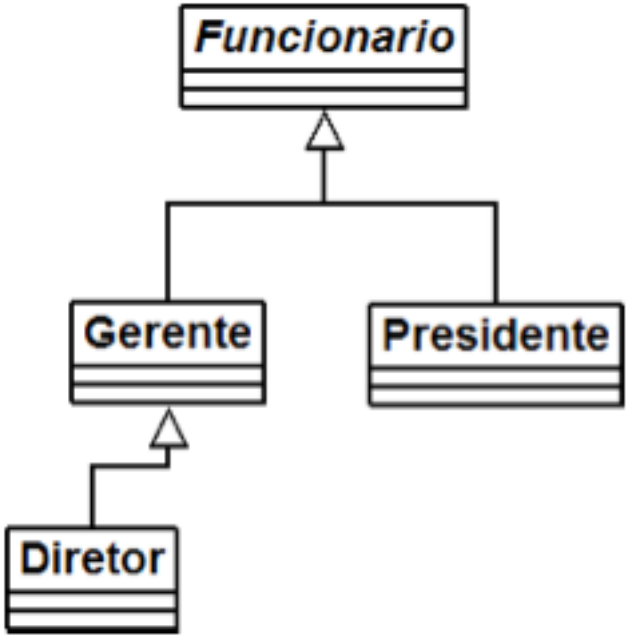
- O código acima **não compila**. O problema é **instanciar** a classe - criar **referência**, você pode. Isso server para criar **polimorfismo**.

Classe abstrata

- Vamos então **herdar** dessa **classe**, **reescrevendo** o **método getBonificacao**:

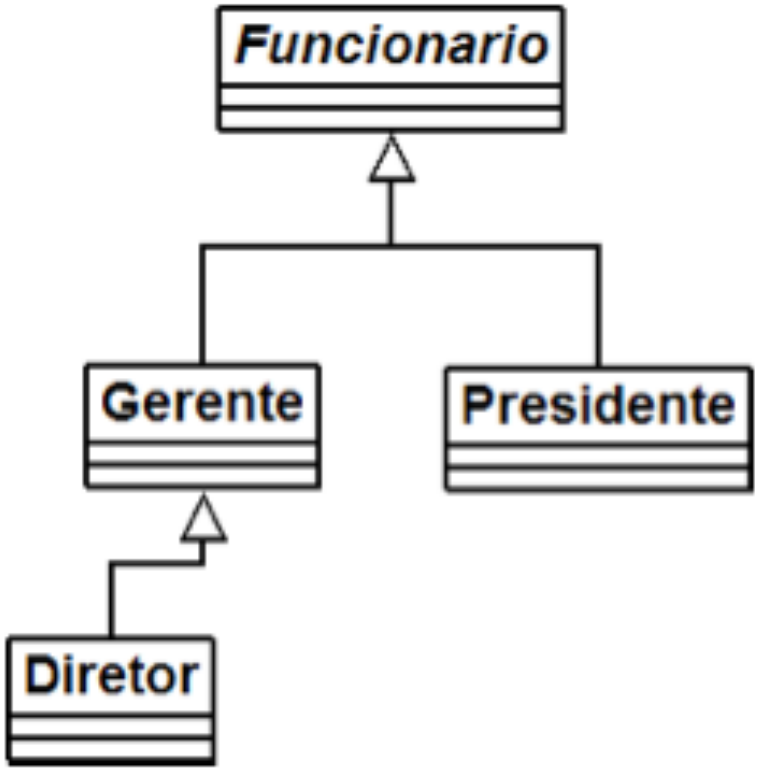
```
class Gerente extends Funcionario {

    public String getBonificacao() {
        return this.salario * 1.4 + 1000;
    }
}
```



Classe abstrata

- Qual é a real **vantagem** de uma **classe abstrata**? Poderíamos ter feito isto com uma **herança comum**.



Métodos abstratos

- Se o **método getBonificacao** não fosse **reescrito**, ele seria **herdado** da classe **mãe**, fazendo com que devolvesse o **salário** mais **20%**.

- Cada **funcionário** em nosso **sistema** tem uma **regra** totalmente **diferente** para ser **bonificado**, faz algum **sentido** ter esse **método** na **classe Funcionario**?

- Poderíamos, **então**, jogar **fora** esse **método** da classe **Funcionario**? O **problema** é que, se ele não **existisse**, não **poderíamos** chamar o **método** apenas com uma **referência** a um **Funcionario**, pois ninguém garante que essa **referência** aponta para um **objeto** que possui esse **método**.

Métodos abstratos

- Existe um **recurso** em Java que, em que um determinado método será **sempre** escrito pelas classes filhas. Isto é, um **método abstrato**.

- Ele **indica** que todas as classes **filhas** (**concretas**)devem **reescrever** esse **método** ou não **compilarão**. É como se você **herdasse** a **responsabilidade** de ter aquele **método**.

```
abstract class Funcionario {

    abstract double getBonificacao();

    // outros atributos e metodos

}
```

Métodos abstratos

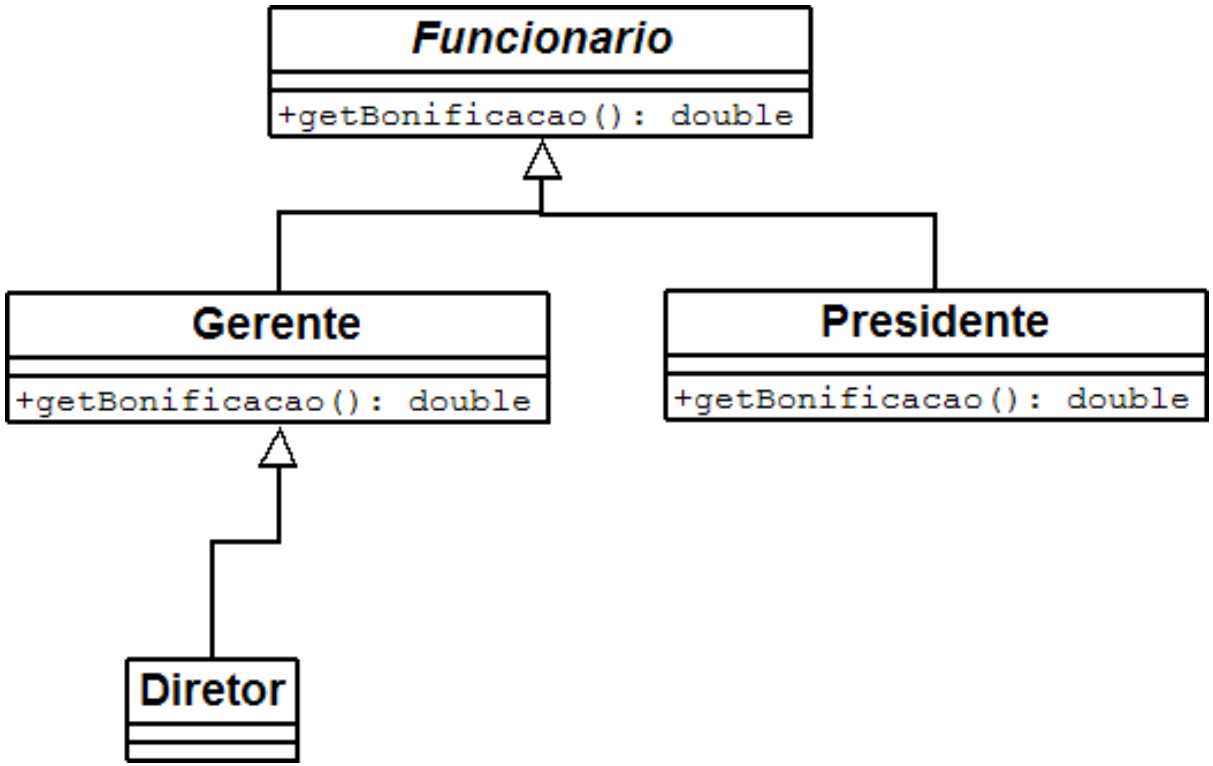
- Qualquer classe que **estender** a classe **Funcionario** será **obrigada** a **reescrever** este **método**, tornando-o "**concreto**". Se não **reescreverem** esse **método**, um **erro** de **compilação** ocorrerá.

```
public void registra(Funcionario f) {
    System.out.println("Adicionando bonificacao do funcionario: " + f);
    this.totalDeBonificacoes += f.getBonificacao();
}
```

- Como posso acessar o método **getBonificacao** se ele não **existe** na classe **Funcionario**?
- Já que o método é **abstrato**, **com certeza** suas **subclasses** têm esse **método**, o que garante que essa **invocação** de **método** não vai **falhar**.

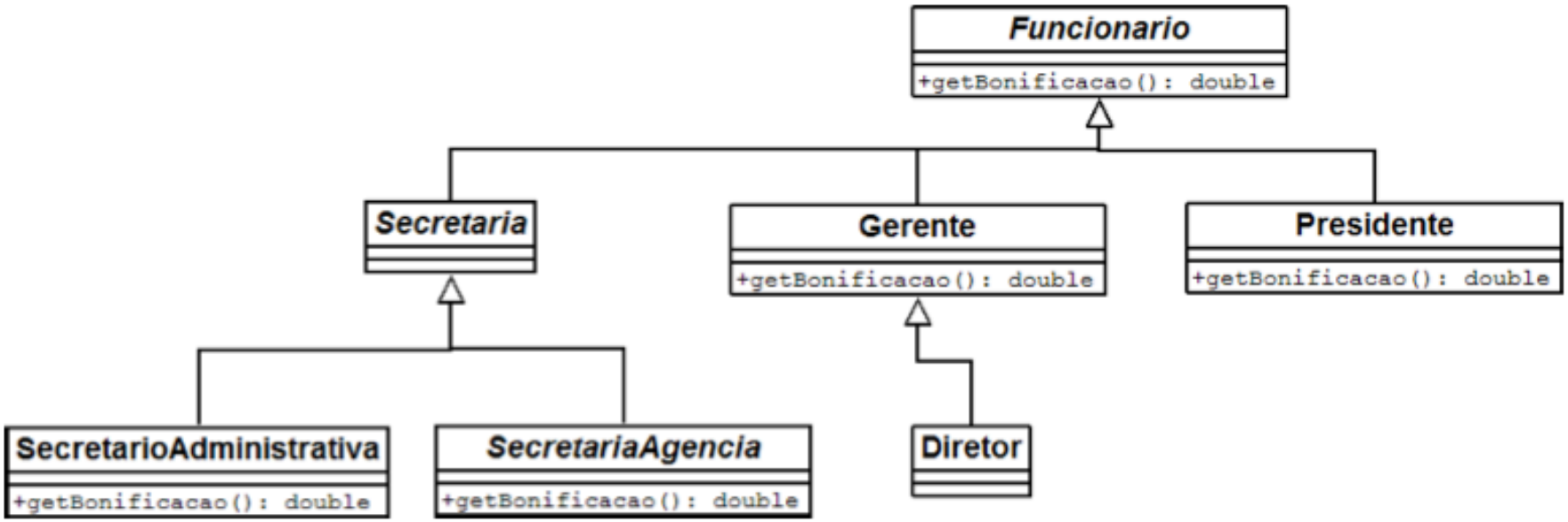
Aumentando Exemplo

- E se, no nosso exemplo de **empresa**, **tivéssemos** o seguinte **diagrama** de classes com os seguintes **métodos**:



- Essas classes vão **compilar**? Vão **rodar**?

Aumentando Exemplo



- Essas classes vão **compilar**? Vão **rodar**?

Operador instanceof

- O **Java** oferece o **operador** **instanceof** como um **mecanismo** para **determinar** em **runtime**, a clase à qual um **objeto** pertence (Isso **normalmente** é **chamado** tipo de **runtime** de objeto);

- Como outros **recursos** da **linguagem**, **instanceof** geralmente é **mal utilizado**;

- Você pode **evitar** muitos **mal-usos** comuns de **instanceof** usando o **polimorfismo**.

Operador instanceof

- Exemplo de uso:

```
view plain  copy to clipboard  print  ?
01.  class Tree{}
02.  class Pine extends Tree{}
03.  class Oak extends Tree{}
04.  public class p32
05.  {  public static void main( String[] args )
06.     {  Tree tree = new Pine();
07.
08.        if( tree instanceof Pine )
09.        System.out.println( "Pine" );
10.
11.        if( tree instanceof Tree )
12.        System.out.println( "Tree" );
13.
14.        if( tree instanceof Oak )
15.        System.out.println( "Oak" );
16.
17.        else System.out.println( "Oops" );
18.     }
19.  }
```

Operador instanceof

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Suponha que você tenha a tarefa de escrever classes para um sistema de folha de pagamento em sua empresa. Considere a seguinte hierarquia de classes e código:

```
interface Employee {
  public int salary();
}


class Manager implements Employee {
  private static final int mgrSal = 40000;
  public int salary() {
    return mgrSal;
  }
}

class Programmer implements Employee {
  private static final int prgSal = 50000;
  private static final int prgBonus = 10000;
  public int salary() {
    return prgSal;
  }

  public int bonus() {
    return prgBonus;
  }
}
```

```
class Payroll {
  public int calcPayroll(Employee emp) {
    int money =  emp.salary();
    if(emp instanceof Programmer)
      money += ((Programmer)emp).bonus(); //Calcula o bônus
    return money;
  }

  public static void main(String arg[]) {
    Payroll pr = new Payroll();
    Programmer prg new Programmer();
    Manager mgr = new Manager();
    System.out.println("Payroll for Programmer is " +
           pr.calcPayroll(prg));

    System.out.println("Payroll for Manager is " +
              pr.calcPayroll(mgr));
  }
}
```

Exercício