



#R4E

Software Developer

Programação Orientada a Objetos

Aula 06 - Herança

Vitor Santos



Conteúdos

1. Palavras Reservadas
2. Herança
3. Exemplos
4. Super
5. Casting Objects
6. Palavras Reservadas Usadas
7. Links Úteis

Palavras Reservadas


| | | | | |
|-----------|----------------|-------------------|--------------|--------------|
| abstract | continue | for | new | switch |
| assert*** | default | goto* | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum**** | <u>instanceof</u> | return | transient |
| catch | <u>extends</u> | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp** | volatile |
| const* | float | native | <u>super</u> | while |

*** not used**

**** added in 1.2**

***** added in 1.4**

****** added in 5.0**



```
public class ProgramadorBackEnd {  
    private String nome, empresaAtual;  
    private int anosExperiência;  
    private double salario;  
    private ArrayList<String> skills;  
    private boolean teamLeader;  
  
    public ProgramadorBackEnd(String nome, String empresaAtual, int anosExperiência,  
double salario, boolean teamLeader) {  
        this.nome = nome;  
        this.empresaAtual = empresaAtual;  
        this.anosExperiência = anosExperiência;  
        this.salario = salario;  
        this.teamLeader = teamLeader;  
        this.skills = new ArrayList<String>();  
    }  
  
    public void mudarEmpresa(String empresaAtual) {  
        this.empresaAtual = empresaAtual;  
    }  
}
```

```
public void setSalario(double salario) {  
    this.salario = salario;  
}  
  
public void aumentarExperiencia() {  
    this.anosExperiência++;  
}  
  
public void adicionarSkill (String novaSkill){  
    this.skills.add(novaSkill);  
}  
  
public void promover(){  
    this.teamLeader = true;  
    this.salario *= 1.2;  
}  
  
public void programarAPI(){  
    // ...  
    System.out.println("API Programada");  
}  
}
```



C

ProgramadorBackEnd

```
String nome;  
String empresaAtual;  
double salario;  
ArrayList<String> skills;  
boolean teamLeader;
```

```
public ProgramadorBackEnd(//argumentos do construtor)  
public void mudarEmpresa(String empresaAtual)  
public void setSalario(double salario)  
public void aumentarExperiencia()  
public void adicionarSkill (String novaSkill)  
public void promover()  
public void programarAPI()
```

```
public class ProgramadorFrontEnd {  
  
    private String nome, empresaAtual;  
    private int anosExperiência;  
    private double salario;  
    private ArrayList<String> skills;  
  
    public ProgramadorFrontEnd(String nome, String empresaAtual, int anosExperiência,  
                                double salario) {  
        this.nome = nome;  
        this.empresaAtual = empresaAtual;  
        this.anosExperiência = anosExperiência;  
        this.salario = salario;  
        this.skills = new ArrayList<String>();  
    }  
  
    public void mudarEmpresa(String empresaAtual) {  
        this.empresaAtual = empresaAtual;  
    }  
}
```



```
public void setSalario(double salario) {  
    this.salario = salario;  
}  
  
public void aumentarExperiencia() {  
    this.anosExperiência++;  
}  
  
public void adicionarSkill (String novaSkill){  
    this.skills.add(novaSkill);  
}  
  
public void desenvolverWebsite(){  
    // ...  
    System.out.println("Website Desenvolvido");  
}  
}
```



C

ProgramadorFrontEnd

```
String nome;  
String empresaAtual;  
double salario;  
ArrayList<String> skills;
```

```
public ProgramadorFrontEnd(//argumentos do construtor)  
public void mudarEmpresa(String empresaAtual)  
public void setSalario(double salario)  
public void aumentarExperiencia()  
public void adicionarSkill (String novaSkill)  
public void desenvolverWebsite()
```

```
public class ProgramadorMobile {  
  
    private String nome, empresaAtual;  
    private int anosExperiência;  
    private double salario;  
    private ArrayList<String> skills;  
    private boolean android;  
  
    public ProgramadorMobile(String nome, String empresaAtual, int anosExperiência,  
                             double salario, boolean android) {  
  
        this.nome = nome;  
        this.empresaAtual = empresaAtual;  
        this.anosExperiência = anosExperiência;  
        this.salario = salario;  
        this.android = android;  
        this.skills = new ArrayList<String>();  
    }  
  
    public void mudarEmpresa(String empresaAtual) {  
        this.empresaAtual = empresaAtual;  
    }  
}
```



```
public void setSalario(double salario) {  
    this.salario = salario;  
}  
  
public void aumentarExperiencia() {  
    this.anosExperiência++;  
}  
  
public void adicionarSkill (String novaSkill){  
    this.skills.add(novaSkill);  
}  
  
public void trocarAndroidParaIOS(){  
    this.android= false;  
    this.salario *= 1.15;  
}  
  
public void programarAppMobile(){  
    // ...  
    System.out.println("App Mobile Programada");  
}  
}
```



C

ProgramadorMobile

```
String nome;  
String empresaAtual;  
double salario;  
ArrayList<String> skills;  
boolean android;
```

```
public ProgramadorMobile(//argumentos do construtor)  
public void mudarEmpresa(String empresaAtual)  
public void setSalario(double salario)  
public void aumentarExperiencia()  
public void trocarAndroidParaIOS()  
public void adicionarSkill (String novaSkill)  
public void desenvolverAppMobile()
```



| <div><div>c</div><div><u>ProgramadorBackEnd</u></div></div> | <div><div>c</div><div><u>ProgramadorFrontEnd</u></div></div> | <div><div>c</div><div><u>ProgramadorMobile</u></div></div> |
|---|--|---|
| <div>String nome; String empresaAtual; double salario; ArrayList<String> skills; boolean teamLeader;</div> | <div>String nome; String empresaAtual; double salario; ArrayList<String> skills;</div> | <div>String nome; String empresaAtual; double salario; ArrayList<String> skills; boolean android;</div> |
| <div>public ProgramadorBackEnd(//argumentos do construtor) public void mudarEmpresa(String empresaAtual) public void setSalario(double salario) public void aumentarExperiencia() public void adicionarSkill (String novaSkill) public void promover() public void programarAPI()</div> | <div>public ProgramadorFrontEnd (//argumentos do construtor) public void mudarEmpresa(String empresaAtual) public void setSalario(double salario) public void aumentarExperiencia() public void adicionarSkill (String novaSkill) public void desenvolverWebsite()</div> | <div>public ProgramadorMobile (//argumentos do construtor) public void mudarEmpresa(String empresaAtual) public void setSalario(double salario) public void aumentarExperiencia() public void trocarAndroidParaIOS() public void adicionarSkill (String novaSkill) public void desenvolverAppMobile()</div> |

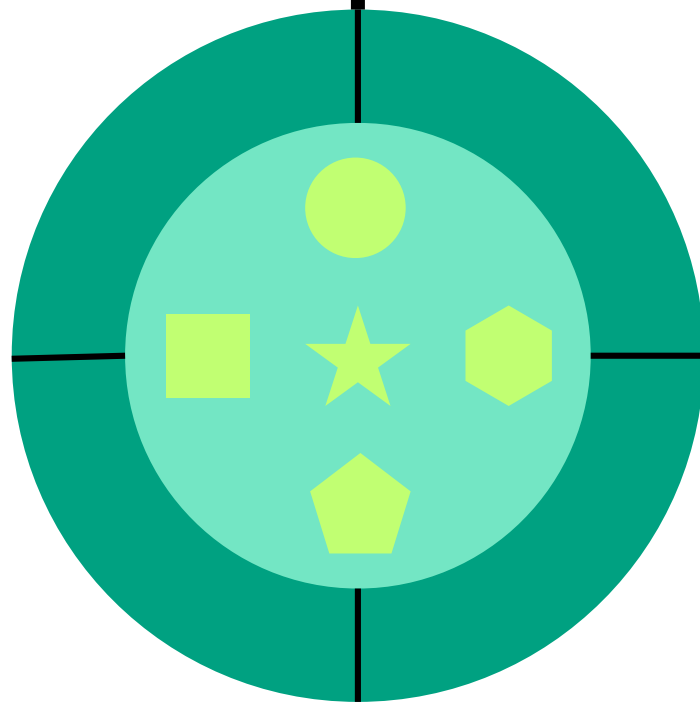
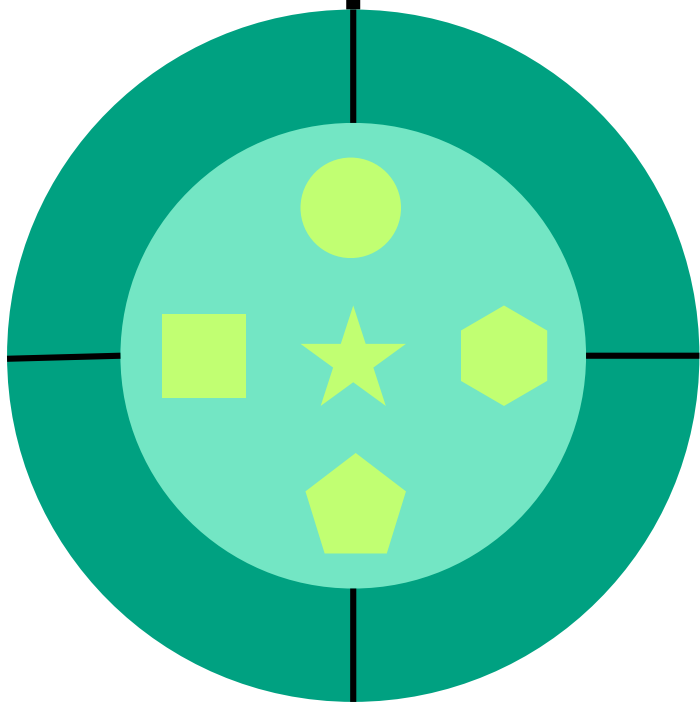
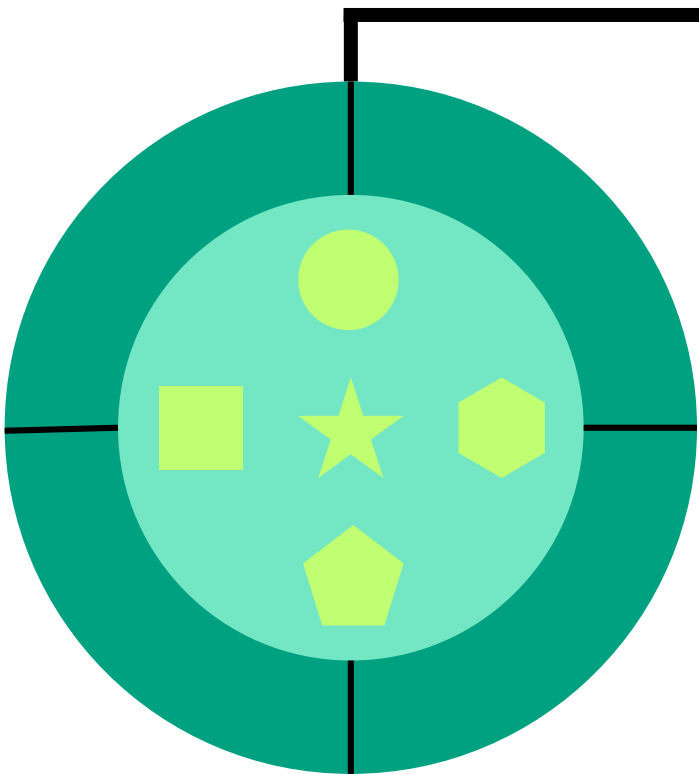
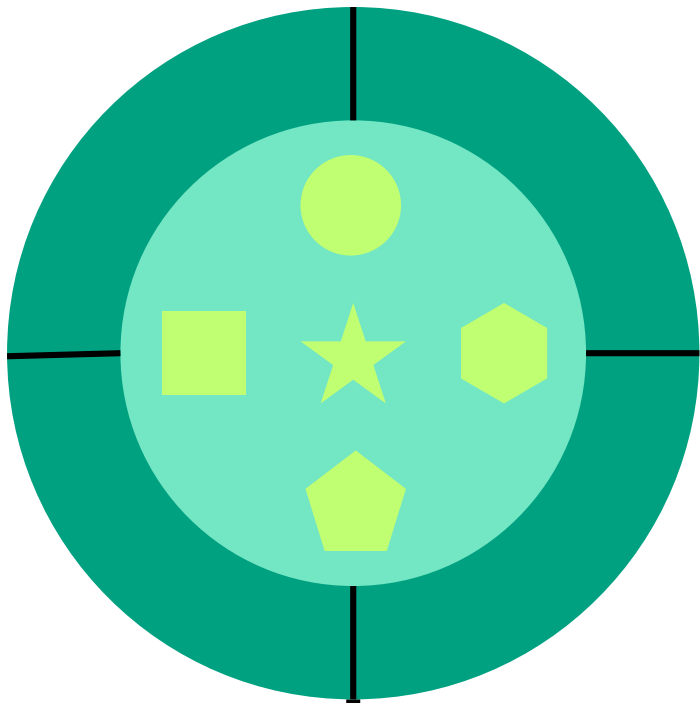
O que é a herança?

- Objetos diferentes têm por vezes bastantes semelhanças uns com os outros.
- Por exemplo, programadores back-end, programadores front-end e programadores mobile partilham todos as mesmas características (nome, empresaAtual, salario, skills).



- No entanto cada uma delas define características adicionais que a diferencia das outras:
 - **Programadores Back-End** podem ser ou não Team Leaders e podem programar uma API.
 - **Programadores Front-End** podem desenvolver um Website.
 - **Programadores Mobile** podem ou não desenvolver em Android e podem programar um App Mobile.
- A programação orientada a objectos permite que as classes herdem estados e comportamentos que são comuns de outras classes.

Programador



ProgramadorBackEnd

ProgramadorFrontEnd

ProgramadorMobile

- Na figura anterior, **Programador** torna-se super classe de **Programador BackEnd**, **Programador FrontEnd** e **Programador Mobile**.
- Em java cada classe só pode ter uma super classe direta e um número ilimitado de sub-classes.



A sintaxe de criação de uma sub-classe é simples:

- Na declaração da classe usar a palavra reservada `extends` seguida do nome da classe que irá herdar.
- Para o caso de Programador Backend:

```
class ProgramadorBackend extends Programador {  
    // atributos e métodos únicos para ProgramadorBackend  
}
```





```
public class Programador {
```

```
// Atributos comuns a todos os Programadores
```

```
private String nome, empresaAtual;
```

```
private int anosExperiência;
```

```
private double salario;
```

```
private ArrayList<String> skills;
```

```
// Método construtor para um Programador
```

```
public Programador(String nome, String empresaAtual,  
                    int anosExperiência, double salario) {
```

```
    this.nome = nome;
```

```
    this.empresaAtual = empresaAtual;
```

```
    this.anosExperiência = anosExperiência;
```

```
    this.salario = salario;
```

```
    this.skills = new ArrayList<String>();
```

```
}
```

// Métodos comuns a todos os Programadores

```
public void mudarEmpresa(String empresaAtual) {  
    this.empresaAtual = empresaAtual;  
}
```

```
public void setSalario(double salario) {  
    this.salario = salario;  
}
```

```
public void aumentarExperiencia() {  
    this.anosExperiência++;  
}
```

```
public void adicionarSkill (String novaSkill){  
    this.skills.add(novaSkill);  
}
```

```
}
```

| <div><div>c</div><div><u>Programador</u></div></div> |
|--|
| <div>String nome; String empresaAtual; double salario; ArrayList<String> skills;</div> |
| <div>public Programador(//argumentos do construtor) public void mudarEmpresa(String empresaAtual) public void setSalario(double salario) public void aumentarExperiencia() public void adicionarSkill (String novaSkill)</div> |

```
public class ProgramadorBackEnd extends Programador {  
  
    // Atributos específicos para ProgramadorBackEnd  
    private boolean teamLeader;  
  
    // Método construtor para ProgramadorBackEnd  
    public ProgramadorBackEnd(String nome, String empresaAtual, int anosExperiência,  
double salario, boolean teamLeader) {  
        super(String nome, String empresaAtual, int anosExperiência, double salario);  
        this.teamLeader = teamLeader;  
    }  
  
    // Métodos específicos para ProgramadorBackEnd  
    public void promover(){  
        this.teamLeader = true;  
        this.salario *= 1.2;  
    }  
  
    public void programarAPI(){  
        // ...  
        System.out.println("API Programada");  
    }  
}
```



C

Programador

```
String nome;  
String empresaAtual;  
double salario;  
ArrayList<String> skills;
```

```
public Programador(//argumentos do construtor)  
public void mudarEmpresa(String empresaAtual)  
public void setSalario(double salario)  
public void aumentarExperiencia()  
public void adicionarSkill (String novaSkill)
```



C

ProgramadorBackend

```
boolean teamLeader;
```

```
public ProgramadorBackend(//argumentos do construtor)  
public void programarAPI()
```

Super

- A sintaxe para invocar o método construtor de uma superclasse é a seguinte:

```
super();
```

```
-- ou --
```

```
super (lista de parâmetros);
```

- Com **super()** é invocado o método construtor sem argumentos enquanto que com **super(lista de parâmetros)** é invocado o construtor com a assinatura correspondente à lista de parâmetros passado para o super.
- O **super()** deve ser invocado na **primeira linha** do método **construtor**.



- Para o exemplo do Programador o construtor de Programador é o seguinte:

```
public Programador(String nome, String empresaAtual,  
                    int anosExperiência, double salario) {  
    this.nome = nome;  
    this.empresaAtual = empresaAtual;  
    this.anosExperiência = anosExperiência;  
    this.salario = salario;  
    this.skills = new ArrayList<String>();  
}
```

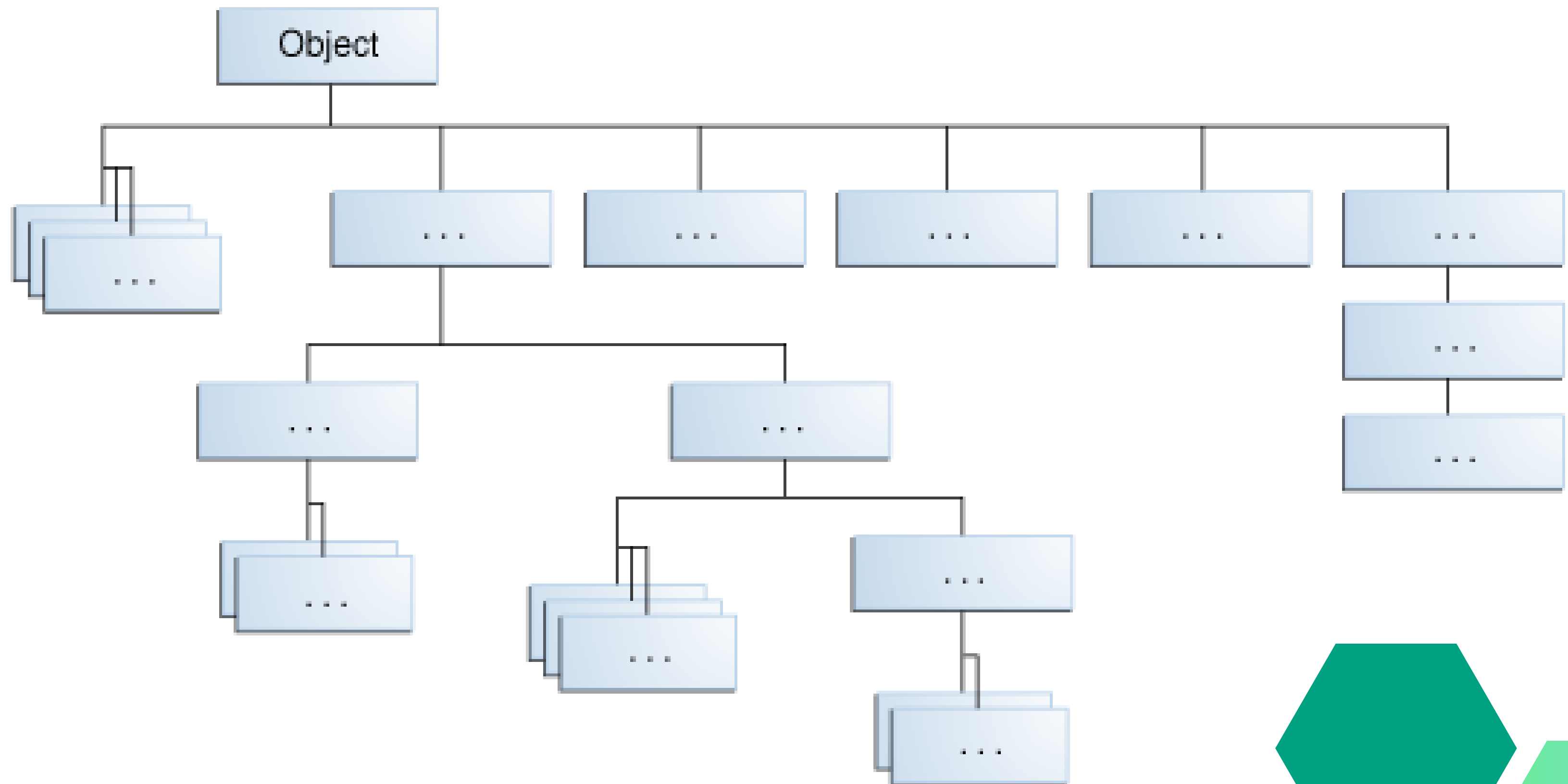
- De destacar que a primeira linha do construtor ProgramadorBackEnd é **super** com a respectiva lista de parâmetros:

```
public ProgramadorBackEnd(String nome, String empresaAtual,  
                           int anosExperiência, double salario,  
                           boolean teamLeader) {  
    super(String nome, String empresaAtual, int anosExperiência,  
           double salario);  
    this.teamLeader = teamLeader;  
}
```

Hierarquia de Classes da Plataforma Java

- A classe **Object** definida no package `java.lang`, define e implementa comportamentos comuns a todas as classes (incluído as criadas por nós).
- Na plataforma java muitas classes derivam directamente de `Object` enquanto que outras derivam dessas classes... formando a hierarquia de classes.





Casting Objects

- Temos visto que o tipo de dados de um objeto é da classe com que foi instanciado. Por exemplo:

```
public ProgramadorBackEnd vitor = new ProgramadorBackEnd(...);
```

- **vitor** é do tipo ProgramadorBackEnd



- Como ProgramadorBackEnd é um descendente de Programador e Object um ProgramadorBackEnd é sempre do tipo Programador e Object.
- O contrário já não se pode afirmar: Programador pode não ser ProgramadorBackEnd. O mesmo acontece com Object

```
Object obj = new ProgramadorBackEnd(...);
```

- **obj** é do tipo Object e ProgramadorBackEnd



- Fazendo o seguinte:

```
ProgramadorBackEnd vitor = obj ;
```

- Como para o compilador a expressão anterior não é verdadeira temos que lhe dizer explicitamente que o objeto em Object irá ser do tipo ProgramadorBackEnd.

```
ProgramadorBackEnd vitor = (ProgramadorBackEnd ) obj ;
```

- Podemos fazer um teste lógico para determinar se a instância é de um determinado tipo:

```
if ( obj instanceof ProgramadorBackEnd ) {  
    ProgramadorBackEnd vitor = (ProgramadorBackEnd) obj ;  
}
```


Palavras Reservadas

| | | | | |
|-----------|----------|------------|------------|--------------|
| abstract | continue | for | new | switch |
| assert*** | default | goto* | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum**** | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp** | volatile |
| const* | float | native | super | while |

*** not used**

**** added in 1.2**

***** added in 1.4**

****** added in 5.0**