

Reuso de Código em LPOO

MATA 55

Rita Suzana



Reuso de Código

- Mecanismo de Reuso:
 - Diminui a necessidade de re-escrever código:
 - menos trabalho para o programador.
 - Permite o aproveitamento de código pré-existente (livre de erro e otimizado):
 - menos chances de cometer erros
 - Em linguagens convencionais: biblioteca de funções e procedimentos



Reuso de classes em POO

- Para criar uma classe que se aproveita de características de uma outra classe
- Delegação (composição)
Uma instância da classe existente é usada como componente da nova classe.
- Herança
A nova classe é uma extensão da classe existente



Composição

- Uma classe possui como atributo uma outra classe.
 - Reuso de atributos e métodos
 - Delegamos a execução dos métodos



Reuso de código

- Reuso de classes em POO:

Exemplo: Um aluno de um curso universitário é modelado pela classe `RegistroAcadêmico`.

- Delegação (composição)

A classe `RegistroAcadêmico` possui o campo `dataDeNascimento`, que delega à classe `Data` a função de armazenar e manipular adequadamente a data de nascimento do aluno.

Delegação ou Composição

```
class RegistroAcademicoDeGraduacao
```

```
{  
    private String nomeDoAluno;  
    private Data dataDeNascimento;  
    private int númeroDeMatrícula;
```

delegação: campos que são instâncias de classes. São manipulados da mesma forma que tipos nativos

```
RegistroAcademicoDeGraduacao(String n, Data d, int m) {  
    nomeDoAluno = n;  
    dataDeNascimento = d;  
    númeroDeMatrícula = m;  
}
```

Chamada implícita do método toString da classe Data. A classe

RegistroAcademicoDeGraduacao delega à classe Data a formatação de seus dados.

```
public String toString() {  
    String resultado = "";  
    resultado += "Matrícula: " + númeroDeMatrícula + " Nome: " + nomeDoAluno + "\n";  
    resultado += "Data de Nascimento: " + dataDeNascimento + "\n";  
    return resultado;  
}  
} // fim da classe RegistroAcademicoDeGraduacao
```

RegistroAcademicoDeGraduacao.java



Delegação ou Composição

```
class DemoRegistroAcademicoDeGraduacao
{
    public static void main(String args[])
    {
        Data nascimento = new Data((byte)10,(byte)4,(short)1940);

        RegistroAcademicoDeGraduacao millôr =
            new RegistroAcademicoDeGraduacao("Millôr Fernandes", nascimento,
            34990917);

        System.out.println(millôr);
    }

} // fim da classe DemoRegistroAcademicoDeGraduacao
```

DemoRegistroAcademicoDeGraduacao.java



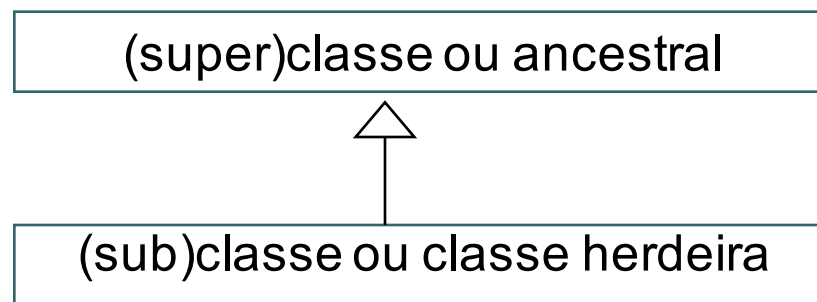
Reuso de Código

○ Herança

- A capacidade de uma classe definir o seu comportamento e sua estrutura aproveitando definições de outra classe
 - Classe **base, super classe**, ou classe **pai**.

Herança

Relacionamento hierárquico entre classes:



A subclasse *herda* da classe:

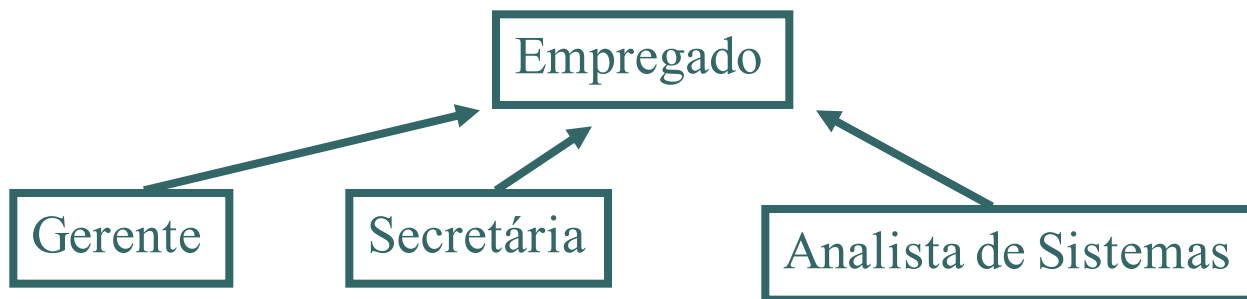
- todos os campos
- todos os métodos

mais especializada

A subclasse pode conter atributos e métodos adicionais

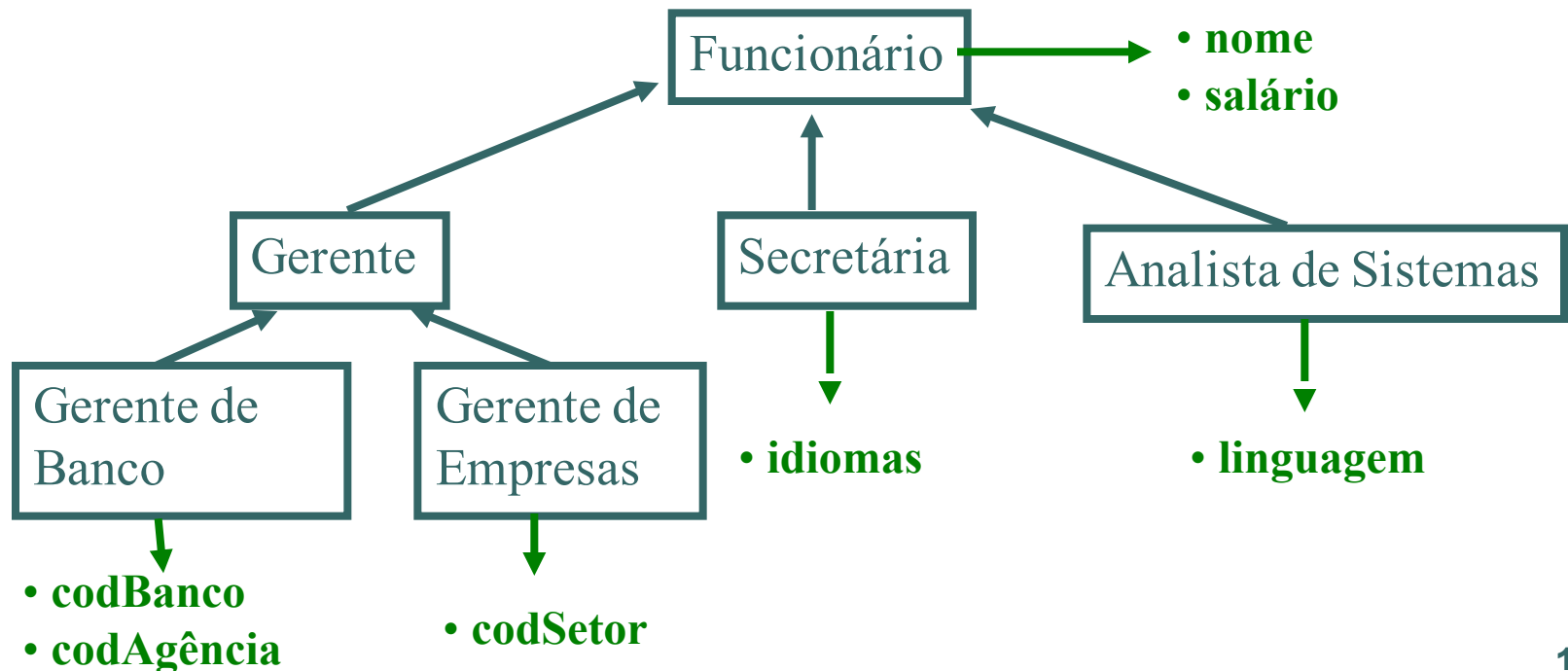
Herança

- A herança não precisa ser interrompida na derivação de uma camada de classes. A coleção de todas as classes que se estendem de um pai comum chama-se **hierarquia de herança**.
- A palavra reservada **extends** é que define que uma classe está herdando de outra.



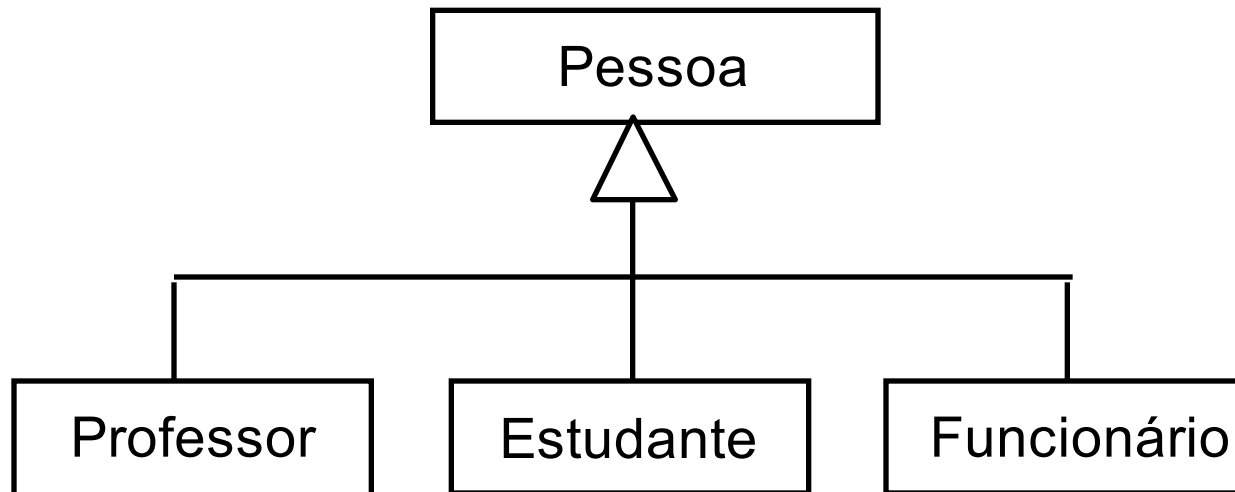
Herança

- A partir destas especificações genéricas podemos construir novas classes, mais específicas, que acrescentem novas características e comportamentos aos já existentes (**Especialização**).
- Através do mecanismo de herança é possível definirmos classes genéricas que agreguem um conjunto de definições comuns a um grande número de objetos (**Generalização**).





Herança





Herança X Composição

- Delegação: **tem-um**
 - Quando se quer as características de uma classe, mas não seus campos e métodos;
 - O *componente* auxilia na implementação da funcionalidade da classe.
- Herança: **é-um**
 - Além de usar as características da superclasse, a subclasse também usa campos e/ou métodos da superclasse



Herança

Pessoa

nome, identidade, nascimento

Pessoa(n,i,nasc);

qualIdentidade();

toString();

Funcionário

admissão, salário

Funcionário(n,i,nasc,adm,sal);

qualSalário();

toString();





Herança

```
class Pessoa {  
    private String nome;  
    private int identidade;  
    private Data nascimento;  
  
    Pessoa(String n,int i,Data d) {  
        nome = n; identidade = i; nascimento = d;  
    }  
  
    public String toString() {  
        return "Nome: "+nome+"\nIdentidade: "+identidade+" "+ "\nData de Nascimento:  
        "+nascimento;  
    }  
  
    final public float qualIdentidade() { return identidade; }  
  
} // fim da classe Pessoa
```

classes herdeiras não podem
sobrepôr este código.

Pessoa.java

Herança

```
class Funcionario extends Pessoa {  
    private Data admissão;  
    private float salário;  
  
    Funcionario(String nome,int id,Data nasc,  
                Data adm,float sal) {  
        super(nome,id,nasc);  
        admissão = adm;  
        salário = sal;  
    }  
    @override  
    public String toString() {  
        return super.toString()+"\n"+  
            "Data de admissão: "+admissão+  
            "\n" + "Salário: "+salário;  
    }  
  
    final public float qualSalário() { return salário; }  
} // fim da classe Funcionario
```

especifica a herança.

nome e idade são privados em Pessoa: mesmo nas subclasses devem ser acessados através dos serviços oferecidos.

O construtor desta classe delega ao construtor da superclasse a tarefa de inicializar os dados herdados.


toString desta classe delega a *toString* da superclasse a impressão de seus dados. Sintaxe diferente para invocar construtor ou método da superclasse.

Funcionário.java



Herança

```
public class Empresa {  
    public static void main(String[] args) {  
        float s; int i;  
        Data d1 = new Data((byte)12,(byte)12,(short)1967);  
        Pessoa p = new Pessoa ("Denise", 3454637, d1);  
        Data d2 = new Data((byte)1,(byte)12,(short)1972);  
        Data d3 = new Data((byte)1,(byte)12,(short)2002);  
        i = p.qualIdentidade();  
        Funcionario f1 =  
            new Funcionario ("Rogerio", 93452128 ,d2 ,d3 ,(float)1000.00);  
        s = f1.qualSalario();  
        i = f1.qualIdentidade();  
        System.out.println(f1);  
    }  
}
```



Funcionário herda as
operações de Pessoa

Empresa.java



Herança

Pessoa

nome, identidade, nascimento

Pessoa(n,i,nasc);

toString();

Funcionário

admissão, salário

Funcionário(n,i,nasc,adm,sal);

qualSalário();

toString();

ChefeDeDepartamento

departamento, promoçãoAChefe

ChefeDeDepartamento(n,i,nasc,adm,sal,dep,prom);

qualDepartamento();

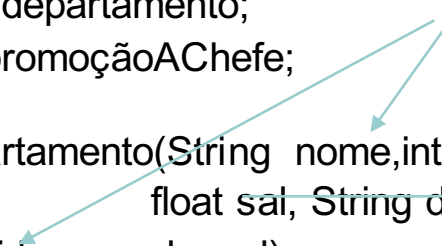
toString();



Herança

```
class ChefeDeDepartamento extends Funcionario {  
  
    private String departamento;  
    private Data promoçãoAChefe;  
  
    ChefeDeDepartamento(String nome,int id,Data nasc, Data adm,  
                           float sal, String dep,Data prom) {  
        super(nome,id,nasc,adm,sal);  
        departamento = dep;  
        promoçãoAChefe = prom; }  
  
    public String toString() {  
        return super.toString()+"\n"+ Departamento:"+departamento+"\n"  
               + "Data de promoção ao cargo:"+promoçãoAChefe; }  
  
    public String qualDepartamento() { return departamento; }  
  
} // fim da classe ChefeDeDepartamento
```

a herança é transitiva



ChefeDeDepartamento.java

Herança

Pessoa
nome, identidade, nascimento
Pessoa(n,i,nasc);
toString();

PacienteDeClínica
planoDeSaúde
PacienteDeClínica(n,i,nasc,plano);
toString();

Funcionário
admissão, salário
Funcionário(n,i,nasc,adm,sal);
qualSalário();
toString();

ChefeDeDepartamento
departamento, promoção
ChefeDeDepartamento(n,i,nasc,adm,sal,dep,prom);
qualDepartamento();
toString();

A herança é uma hierarquia:
da raiz para as folhas;
sem relacionamento entre irmãos.

Não há herança múltipla



Hierarquia de classes em Java

Class Hierarchy

oclass java.lang.**Object**

oclass java.lang.**Boolean** (implements java.io.Serializable)

oclass java.lang.**Character** (implements java.lang.Comparable, java.io.Serializable)

oclass java.lang.**Character.Subset**

oclass java.lang.**Character.UnicodeBlock**

oclass java.lang.**Class** (implements java.io.Serializable)

...

oclass java.lang.**Math**

oclass java.lang.**Number** (implements java.io.Serializable)

oclass java.lang.**Byte** (implements java.lang.Comparable)

oclass java.lang.**Double** (implements java.lang.Comparable)

oclass java.lang.**Float** (implements java.lang.Comparable)

oclass java.lang.**Integer** (implements java.lang.Comparable)

oclass java.lang.**Long** (implements java.lang.Comparable)

oclass java.lang.**Short** (implements java.lang.Comparable)

...



Object: A raíz da hierarquia

- Todas as classes herdam de Object, mesmo que não contenham a declaração de herança.
- Contém apenas métodos genéricos, que devem ser reimplementados pelas classes.

Object: A raiz da hierarquia

Method Summary

protected <u>O</u> <u>b</u> <u>j</u> <u>e</u> <u>c</u> <u>t</u>	<u>clone()</u> Creates and returns a copy of this object.
boolean	<u>equals(Object obj)</u> Indicates whether some other object is "equal to" this one.
protected v oid	<u>finalize()</u> Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
<u>Class</u>	<u>getClass()</u> Returns the runtime class of an object.
int	<u>hashCode()</u> Returns a hash code value for the object.
void	<u>notify()</u> Wakes up a single thread that is waiting on this object's monitor.
void	<u>notifyAll()</u> Wakes up all threads that are waiting on this object's monitor.
<u>String</u>	<u>toString()</u> Returns a string representation of the object.
void	<u>wait()</u> Causes current thread to wait until another thread invokes the <u>notify()</u> method or the <u>notifyAll()</u> method for this object.
void	<u>wait(long timeout)</u> Causes current thread to wait until either another thread invokes the <u>notify()</u> method or the <u>notifyAll()</u> method for this object, or a specified amount of time has elapsed.
void	<u>wait(long timeout, int nanos)</u> Causes current thread to wait until another thread invokes the <u>notify()</u> method or the <u>notifyAll()</u> method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.



Referencia Super

A palavra reservada **super**, nos possibilita:

- referência a classe base (pai) do objeto;
- acesso a métodos que foram redefinidos, ou seja, métodos que possuem a mesma assinatura na classe Pai e na classe derivada;
- utilização pela classe derivada do construtor da classe pai.



Exercício básico de herança:

- a. Construa inicialmente uma classe em Java para representar pessoas. Cada pessoa deverá ter um nome e um endereço.
- b. Em seguida, crie uma classe para representar estudantes e professores. Dado que todo estudante é também uma pessoa, os atributos adicionais de estudante serão nome da escola e a série que cursa.
- c. Como todo professor é também uma pessoa, o atributo adicional então para professor será a disciplina que ele ensina.
- d. Posteriormente, crie uma classe para representar estudantes internacionais, que tem como diferença o atributo referente ao seu país de origem. Para todas as classes, crie métodos acessores (*gets* e *sets*) para obter ou modificar os valores dos atributos.
- e. Instancie no BlueJ objetos diversos de cada classe. Em seguida, verifique os métodos disponíveis para serem chamados. Por exemplo, solicite obter o nome do estudante internacional ou o endereço do professor.



Sobreposição

○ Sobreposição de campos

- Um campo declarado na subclasse sobrepõe o campo de mesmo nome da superclasse.

○ Sobreposição de métodos

- Método herdado pode não ser adequado para os objetos das subclasses.
- Um método declarado na subclasse sobrepõe o método com a mesma assinatura.
- O método sobreposto da superclasse, se não for privado, pode ser invocado através da palavra **super**.



Sobreposição

- O Java, automaticamente reconhece de qual classe um objeto é instanciado
 - se de uma classe filha ou de uma classe pai.
 - caso tenhamos uma sobreposição de métodos (*override*)
 - o Java consegue diferenciar qual o método de qual classe aquele objeto esta se referenciando.



Exemplo

```
class Computador
{
    private boolean ligado =true;

    public void Desligar()
    {
        ligado = false;
    }

    public setLigado( boolean estado)
    { this.ligado=estado;}
}
```

```
class ComputadorSeguro extends Computador {
    private boolean executando = true;

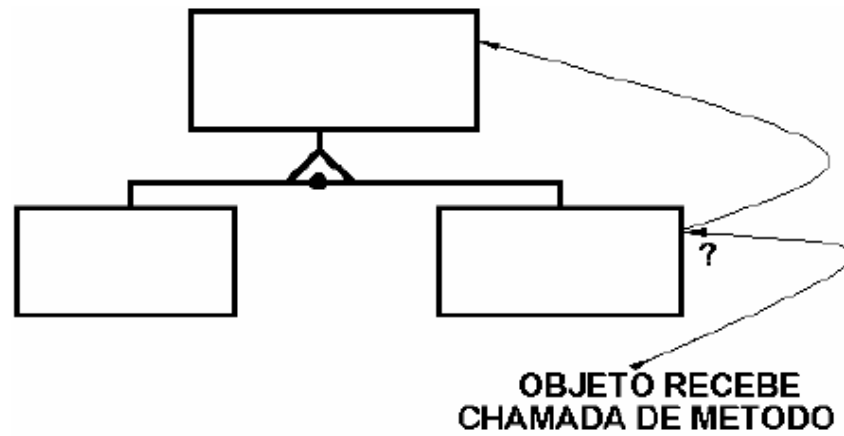
    @override
    public void Desligar()
    {
        if ( executando)
            System.out.println("Há programas rodando. Não
                                desligue!");

        else
            super.setLligado( false);

        /* alternativa usar o metodo desligado*/
    }
}
```

Sobreposição

- A busca da implementação do método a ser executado ocorre de baixo para cima na hierarquia





Sobreposição

- Um método **public** (subclasse) pode sobrepor um método **private** (superclasse);
- Um método **private** (subclasse) não pode sobrepor um método **public** (superclasse);
- Um método estático não pode ser sobreposto;
- Um método **final** é herdado pelas subclasses, mas não pode ser sobreposto.



Sobreposição

- Exercício Animais
 - Criar o modelo e implementar em Java uma classe que represente animais. A classe possui os atributos tipo (mamífero, anfíbio, ave), nome, idade. Esta classe deve possuir o seguinte método:
 - métodos para calcular a quantidade de alimento consumido por dia e por tempo de vida, sabendo-se que a cada ano de vida um mamífero come 2 quilos/dia, uma ave come 100g/dia um anfíbio 20g/dia. A quantidade deve ser retornada em quilogramas.
- Q4listaHerança

Reuso e Herança

1. Campos da superclasse *que não sejam privados* podem ser utilizados diretamente.
2. Campos privados da superclasse devem ser usados através dos serviços oferecidos pela superclasse.
3. Construtores da superclasse podem ser utilizados através da palavra *super*.
4. Somente os Construtores da superclasse imediata podem ser utilizados diretamente.
5. Métodos da superclasse *que não sejam privados* podem ser utilizados diretamente.
6. Métodos sobrepostos da superclasse que não sejam privados (*public ou protected*) podem ser utilizados através da palavra *super*.

```
class Funcionario extends Pessoa {  
    private Data admissão;  
    private float salário;  
  
    Funcionario(String nome,int id,Data  
        nasc,Data adm,float sal) {  
        super(nome,id,nasc); ← (3)  
        admissão = adm;  
        salário = sal;  
    }  
  
    public String toString() { ← (6)  
        return super.toString()+"\n"+  
            "Data de admissão: "+admissão+  
            "\n" + "Salário: "+salário;  
    }  
  
    final public float qualSalário() { return  
        salário; }  
  
} // fim da classe Funcionario  
Funcionário.java
```




Exercício

Exercício básico de herança:

- a. Construa inicialmente uma classe em Java para representar pessoas. Cada pessoa deve ter um nome e um endereço.
- b. Em seguida, crie uma classe para representar estudantes e professores. Dado que todo estudante é também uma pessoa, os atributos adicionais de estudante serão nome da escola e a série que cursa.
- c. Como todo professor é também uma pessoa, o atributo adicional então para professor será a disciplina que ele ensina.
- d. Posteriormente, crie uma classe para representar estudantes internacionais, que tem como diferença o atributo referente ao seu país de origem. Para todas as classes, crie métodos acessores (*gets* e *sets*) para obter ou modificar os valores dos atributos.
- e. Instancie no BlueJ objetos diversos de cada classe. Em seguida, verifique os métodos disponíveis para serem chamados. Por exemplo, solicite obter o nome do estudante internacional ou o endereço do professor.
- f. Se quiséssemos distinguir professores universitários de professores de cursinho, o que faríamos? Realize esta alteração.
- g. Se quiséssemos obter o nome do professor com o pré-fixo “Prof.”, o que faríamos? Por exemplo, se o nome do professor for “Fulano”, obter então o nome dele assim “Prof.Fulano”. Realize esta alteração.

(lista4q1)

● Exemplo

```
public class Automovel  
{ public static final byte movidoAGasolina = 1;  
  public static final byte movidoAAlcool = 2;  
  public static final byte movidoADiesel = 3;  
  private static final byte numeroMaximoDePrestacoes = 24;
```

Classe Pai

Declaração de constantes.

```
private String modelo;  
private String cor;  
private byte combustivel;
```

Atributos

```
public Automovel(String m, String c, byte comb)  
{ setModelo( m);  
  setCor(c);  
  SetCombustivel(comb);  
}
```

Construtor

```
public byte quantasPrestacoes()  
{ return numeroMaximoDePrestacoes;  
}
```

● Exemplo

```
public float quantoCusta()
{
    float preco = 0;
    switch (getCombustivel())
    {
        case movidoAGasolina: preco = 12000; break;
        case movidoAAlcool: preco = 10500; break;
        case movidoADiesel: preco = 11000; break;
    }
    return preco;
}

public String toString()
{
    String resultado;
    resultado = getModelo()+" "+getCor()+"\n";
    switch(getCombustivel())
    {
        case movidoAGasolina: resultado += "Gasolina \n"; break;
        case movidoAAlcool: resultado += "Álcool \n"; break;
        case movidoADiesel: resultado += "Diesel \n"; break;
    }
    return resultado;
}
}
```

● Exemplo

```
public class AutomovelBasico extends Automovel
{ private boolean retrovisorDoLadoDoPassageiro;
  private boolean limpadorDoVidroTraseiro;
  private boolean radioAMFM;
```

A palavra **extends** é que determina que AutomovelBasico herda de Automovel.

Atributos

```
public AutomovelBasico (String m, String c, byte comb, boolean r,
                        boolean l, boolean af)
{ super(m,c,comb);
  setRetrovisorDoLadoDoPassageiro(r);
  setLimpadorDoVidroTraseiro(l);
  setRadioAMFM(af);
}
```

A palavra **super**, indica que deve ser usado o construtor da classe Pai.

```
public AutomovelBasico (String m, String c, byte comb)
{ super(m,c,comb);
  setRetrovisorDoLadoDoPassageiro(true);
  setLimpadorDoVidroTraseiro(true);
  setRadioAMFM(true);
}
```

● Exemplo

```
public float quantoCusta()
{ float preco = super.quantoCusta();
  if (getRetrovisorDoLadoDoPassageiro() == true)
    preco = preco + 280;
  if (getLimpadorDoVidroTraseiro() == true)
    preco = preco + 650;
  if (getRadioAMFM() == true)
    preco = preco + 190;
  return preco;
}

public String toString()
{ String resultado = super.toString();
  if (getRetrovisorDoLadoDoPassageiro() == true)
    resultado += "Com retrovisor do lado direito \n";
  if (getLimpadorDoVidroTraseiro() == true)
    resultado += "Com limpador traseiro \n";
  if (getRadioAMFM() == true)
    resultado += "Com radio \n";
  return resultado;
}}
```

Os dois métodos apresentados nessa transparência possuem a mesma assinatura da classe Automovel, o que caracteriza uma redefinição de métodos da classe Pai.

A palavra **super**, indica que deve ser chamado o método quantoCusta() e toString() da classe Pai.

Exemplo

```
public class DemoAutomovel
{ public static void main(String arg[])
{ Automovel a = new Automovel("Fusca","verde",
Automovel.movidoAAAlcool);

System.out.println(a.toString());
System.out.println(a.quantoCusta());
System.out.println(a.quantasPrestacoes());

AutomovelBasico ab = new AutomovelBasico("Corsa","cinza",
Automovel.movidoAGasolina,true,true,false);

System.out.println(ab.toString());
System.out.println(ab.quantoCusta());
System.out.println(ab.quantasPrestacoes());
}
}
```

Instância de um objeto da classe Automovel.

Instância de um objeto da classe AutomovelBasico.

Observe que o método quantasPrestacoes() está sendo acessado através de um objeto da classe AutomovelBasico. Isso só é possível porque a classe AutomovelBasico herda da classe Automovel, assim todos os atributos e métodos da classe Pai podem ser usados pela classe derivada.