

Linguagem de Programação II

Classes e objetos II



Universidade do Estado do Rio de Janeiro-UERJ
Instituto de Matemática e Estatística-IME
Ciência da Computação
Professor: Alexandre Sztajnberg

Um programa com diversas classes

```
public class Automovel{  
    public String modelo;  
    public int ano;  
    private boolean ligado;  
}  
  
//Instanciar novo automóvel  
public Automovel(String m, int a){  
    modelo = m;  
    ano = a;  
    ligado = false;  
}  
  
//Ligar automóvel  
public void liga(){  
    ligado = true;  
}  
  
//Desligar automóvel  
public void desliga(){  
    ligado = false;  
}  
}
```

Atributos

Métodos



Automovel.java

Um programa com diversas classes

```
class Motorista {
    private String nome;
    private Automovel carro;

    //Instancia novo Motorista
    public Motorista( String n, Automovel a){
        nome = n;
        carro = a;
    }

    //Obter o nome do motorista
    public String obterNome(){
        return nome;
    }

    //Obter o carro do motorista
    public Automovel obterCarro(){
        return carro;
    }
}
```

Atributos

Métodos



Motorista.java

Um programa com diversas classes

```
class MeuPrograma{
    private Automovel carro1;
    private Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carro1);
        motorista2 = new Motorista("Pedro", carro1);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());
    }
}
```

Atributos

Método
(Entry point do programa)



MeuPrograma.jav

Um programa com diversas classes

```
class MeuPrograma{
    private Automovel carrol;
    private Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carrol = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carrol);
        motorista2 = new Motorista("Pedro", carrol);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```

ERRO

Qual o problema? Compile e veja a mensagem de erro!

Tem relação com o contexto “de instância”, “de classe” e “local”

A solução depende do que o programa precisa ...

MeuPrograma.jav

Um programa com diversas classes (solução 1)

```
class MeuPrograma{
    private Automovel carrol;
    private Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carrol = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carrol);
        motorista2 = new Motorista("Pedro", carrol);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```

Já que ninguém se manifestou em sala ...

Solução 1, tem haver com “dar uma solução rápida” e manter tudo num contexto estático ...

MeuPrograma.jav

Um programa com diversas classes (solução 2)

```
class MeuPrograma{

    //Entry point do programa
    public static void main( String args[] ){

        Automovel carrol;
        Motorista motorista1,motorista2;

        //Instanciando novos objetos
        carrol = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carrol);
        motorista2 = new Motorista("Pedro", carrol);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```

Solução 2, também tem haver com "dar uma solução rápida", mas declarar as variáveis localmente (ou seja, não são mais "de instância" ... Veja se isso resolve...

Declarado no método *main* (que é "de classe"), as variáveis existem enquanto o método é executado.



MeuPrograma.java

Um programa com diversas classes (solução 3)

```
class MeuPrograma{
    private Automovel carrol;
    private Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        MeuPrograma mp = new MeuPrograma ();
        mp.executa();

    }

    public void executa() {
        //Instanciando novos objetos
        carrol = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carrol);
        motorista2 = new Motorista("Pedro", carrol);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```

A Solução 3 é mais completa. Se a necessidade é criar variáveis ou atributos de instância, então faça isso!

Mas para isso funcionar (se você entendeu o problema já sabe por que não funciona de outra forma), uma instância tem que existir.

Então, criamos um instância, da própria classe sendo declarada. Isso pode? Pode! ... E se você entendeu "de classe" e de "instância", isso não é mais misterioso...



MeuPrograma.java

Executando o programa

```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;
```

Objeto MeuPrograma

Automovel
+modelo: String
+ano: int
-ligado: boolean
~Automovel()
+void ligar()
+void desligar()

Motorista
-nome: String
-carro:
Automovel
~Motorista()
+String obterNome()
+Automovel obterCarro()

JVM

Executando o programa

```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;
```

Objeto MeuPrograma

carro1

motorista1

motorista2

"null"

"null"

"null"

Automovel
+modelo: String
+ano: int
-ligado: boolean
~Automovel()
+void ligar()
+void desligar()

Motorista
-nome: String
-carro:
Automovel
~Motorista()
+String obterNome()
+Automovel obterCarro()

JVM

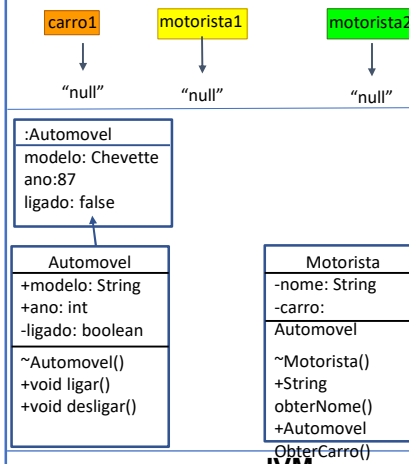
Executando o programa

```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
    }
}
```

Objeto MeuPrograma



JVM

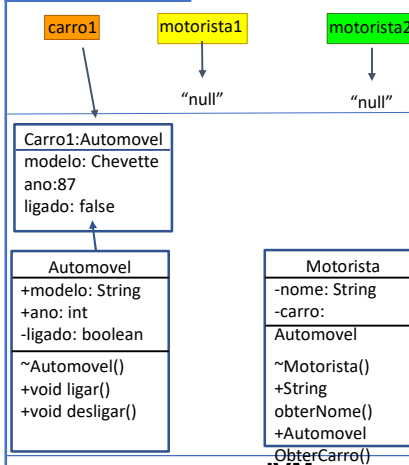
Executando o programa

```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
    }
}
```

Objeto MeuPrograma



JVM

Executando o programa

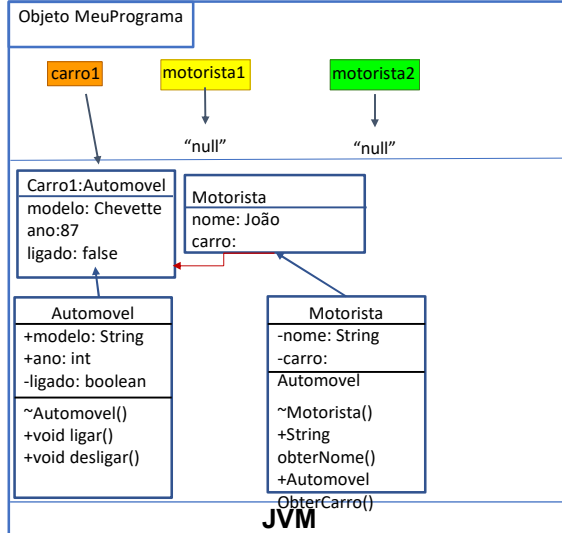
```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carro1);
        motorista2 = new Motorista("Pedro", carro1);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```



Executando o programa

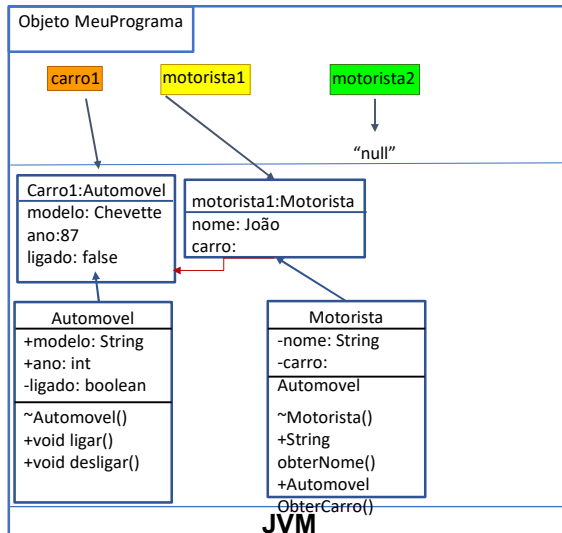
```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carro1);
        motorista2 = new Motorista("Pedro", carro1);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```



Executando o programa

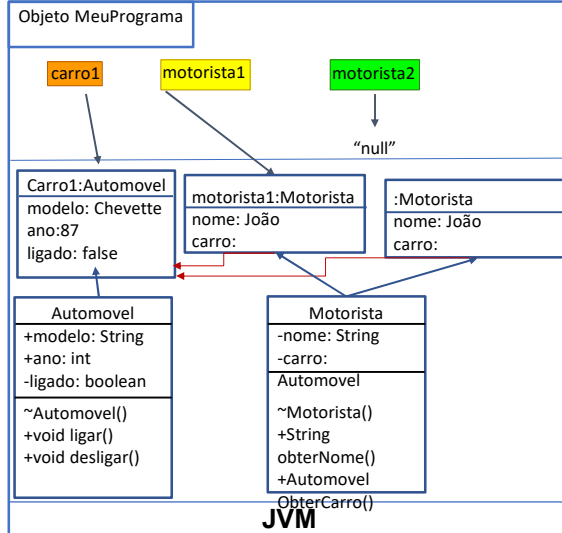
```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carro1);
        motorista2 = new Motorista("Pedro", carro1);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```



Executando o programa

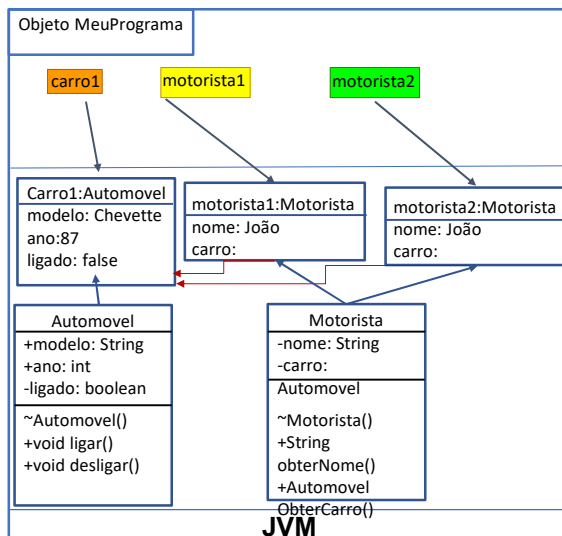
```
class MeuPrograma{
    private static Automovel carro1;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carro1 = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carro1);
        motorista2 = new Motorista("Pedro", carro1);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());

    }
}
```



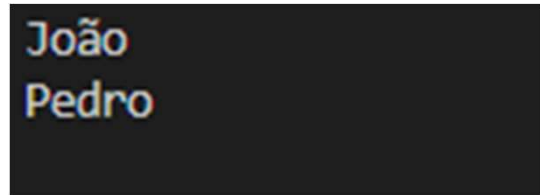
Executando o programa

```
class MeuPrograma{
    private static Automovel carrol;
    private static Motorista motorista1,motorista2;

    //Entry point do programa
    public static void main( String args[] ){

        //Instanciando novos objetos
        carrol = new Automovel("Chevette", 87);
        motorista1 = new Motorista("João", carrol);
        motorista2 = new Motorista("Pedro", carrol);

        //Imprimindo o nome dos motoristas
        System.out.println( motorista1.obterNome());
        System.out.println( motorista2.obterNome());
    }
}
```



João
Pedro

Características e mecanismos importantes

☐ Encapsulamento

- Classes e objetos contendo dados/variáveis/atributos e métodos
- Visibilidade adequada, esconder o que não precisa ser público, deixar público (apenas) o que pode ser usado por objetos externos (clientes?)
- Ser auto-contido, preparado para a modularidade

☐ Herança

- Conceito, mecanismo e sintaxe para facilitar o reuso, permitindo definir uma classe (filha, subclasse, herdeira) a partir de outra classe (mãe/pai, superclasse)
- A herança permite aproveitar atributos e métodos, “como se fosse seu”, criar novos atributos e métodos, aumentando as funcionalidades ou especializando a classe pai
- O encapsulamento deve ser observado em conjunto

☐ Polimorfismo

- Uma instância de objeto de uma classe que participa de uma hierarquia de herança pode ser enxergada de “várias formas”, relacionadas com as superclasses desta hierarquia.

Encapsulamento

- ❑ É um mecanismo de linguagens orientados a objetos que permite que o programador oculte elementos e funcionalidades internas para objetos externos, visibilidade.
- ❑ Objetos externos, que fazem uso de um objeto qualquer vão ter acesso apenas a elementos visíveis, não sendo permitido o acesso a elementos ocultos. Isso garante que outros objetos não irão modificar erroneamente essas funcionalidades, gerando uma maior segurança.
- ❑ Módulos auto-contidos, com todas as funcionalidades relativas ao que a classe representa
- ❑ Outras vantagens:
 - ❑ O programador não precisa conhecer todas as classes internas.
 - ❑ Garante certo grau de independência a classe.
 - ❑ Facilita a modificação posterior do programa.
 - ❑ Facilita o reuso.

Encapsulamento

```
public class Automovel{  
    public String modelo;  
    public int ano;  
    private boolean ligado;  
  
    //Instanciar novo automóvel  
    public Automovel(String m, int a){  
        modelo = m;  
        ano = a;  
        ligado = false;  
    }  
  
    //Ligar automóvel  
    public void liga(){  
        ligado = true;  
    }  
  
    //Desligar automóvel  
    public void desliga(){  
        ligado = false;  
    }  
}
```

Atributos e métodos encapsulados da classe



Automovel.java

Encapsulamento

```
public class Automovel{
    public String modelo;
    public int ano;
    private boolean ligado;

    //Instanciar novo automóvel
    public Automovel(String m, int a){
        modelo = m;
        ano = a;
        ligado = false;
    }

    //Ligar automóvel
    public void liga(){
        ligado = true;
    }

    //Desligar automóvel
    public void desliga(){
        ligado = false;
    }
}
```

Como o campo é privado, classes externas não podem ver nem modificar esse atributo.



Automovel.java

Encapsulamento

```
public class Automovel{
    public String modelo;
    public int ano;

    //Instanciar novo automóvel
    public Automovel(String m, int a){
        modelo = m;
        ano = a;
        ligado = false;
    }

    //Ligar automóvel
    public void liga(){
        ligado = true;
    }

    //Desligar automóvel
    public void desliga(){
        ligado = false;
    }
}
```

Basicamente, a visão externa da classe seria esta. Possuindo somente os atributos modelo e ano.



Automovel.java

Herança

- ❑ <http://152.92.236.11/javatutor/java/landl/subclasses.html>
- ❑ A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).
- ❑ Excepting *Object*, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of *Object*.
- ❑ Classes can be derived from classes that are derived from classes that are derived from classes, and so on, and ultimately derived from the topmost class, *Object*. Such a class is said to be descended from all the classes in the inheritance chain stretching back to *Object*.

Mecanismo de herança

- ❑ The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class.
 - In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.
- ❑ A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. (os membros públicos e protected).
- ❑ Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

O que voce pode fazer numa subclasse?

- ☐ A subclass inherits all of the public and protected members of its parent
- ☐ If the subclass is in the same package as its parent, it also inherits the package-private members of the parent.
- ☐ You can use the inherited members as is, replace, hide, or supplement them with new members:
 - The inherited fields can be used directly, just like any other fields.
 - Declare a field in the subclass with the same name as the one in the superclass, thus hiding it (not recommended).
 - Declare new fields in the subclass that are not in the superclass.
 - The inherited methods can be used directly as they are.
 - Write a new instance method in the subclass that has the same signature as the one in the superclass (overriding).
 - Write a new static method in the subclass that has the same signature as the one in the superclass (hiding)
 - You can declare new methods in the subclass that are not in the superclass.
 - You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

Reengenharia

- ☐ Outro ponto importante: a classificação dos objetos, a hierarquia de herança fica disponível para o programa e deve ser usado.

Automóvel.java

```
class Automóvel {
public String modelo;
public int ano;
private boolean ligado;

// Instanciar novo automóvel
public Automóvel( String m, int a ) {
modelo = m;
    ano = a;
    ligado = false;
}

// Ligar automóvel
public void liga() {
    ligado = true;
}

// Desligar automóvel
public void desliga() {
    ligado = false;
}
}
```



Automóvel.java

```
class Automóvel {
    public int ano;
    private boolean ligado;

    // Instanciar novo automóvel
    public Automóvel( int a ) {
        ano = a;
        ligado = false;
    }

    // Ligar automóvel
    public void liga() {
        ligado = true;
    }

    // Desligar automóvel
    public void desliga() {
        ligado = false;
    }
}
```

Chevette.java

```

class Chevette
    extends Automóvel {

    // Instanciar novo Chevette:
    public Chevette( int a ) {

        // Acessando o construtor da
        // classe Automóvel:
        super(a);

        // Consertar o Chevette:
        public void conserta( ) {
            ligado = false;
        }
    }
        
```

Operação inválida!!!

O Atributo "ligado" é privado na classe Automóvel. Logo, a classe Chevette não pode acessá-lo mesmo sendo sua herdeira.

Opções

```

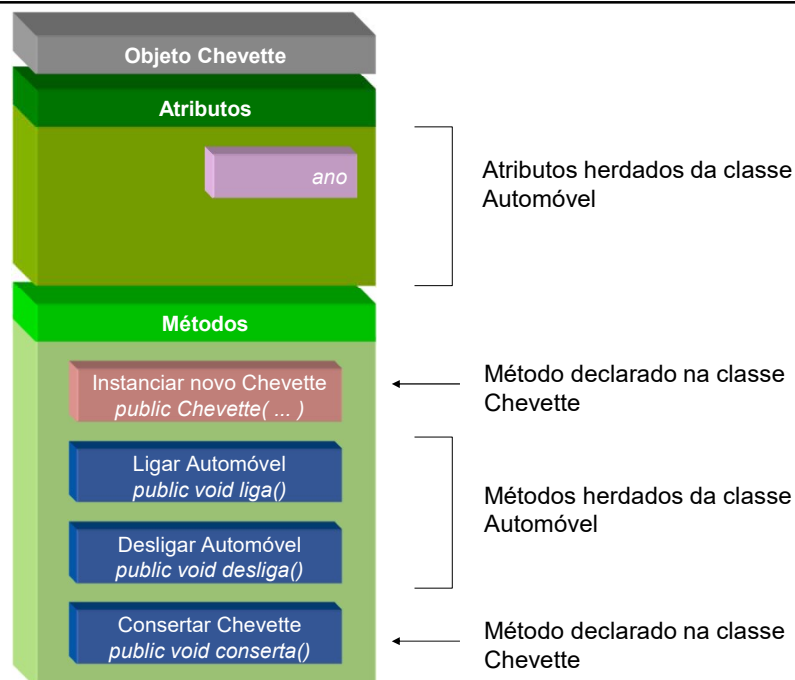
// Consertar o Chevette:
public void conserta( ) {

    //desliga();
    //this.desliga();
    super.desliga();

}
        
```

Private Members in a Superclass

- ❑ A subclass does not inherit the private members of its parent class.
 - However, if the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.
- ❑ A nested class has access to all the private members of its enclosing class—both fields and methods. Therefore, a public or protected nested class inherited by a subclass has indirect access to all of the private members of the superclass.



Operações válidas com um objeto Chevette:

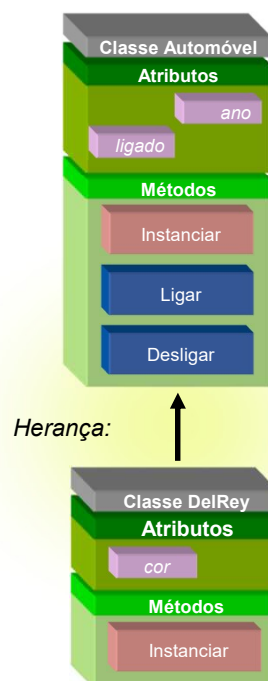
```
// Instanciando um novo Chevette:
Chevette meuChevette = new Chevette( 88 );

// Acessando o método declarado na classe
Chevette:
meuChevette.conserta( );

// Acessando os métodos declarados na super
// classe Automóvel:
meuChevette.liga( );
meuChevette.desliga( );

// Acessando os atributos declarados na super
// classe Automóvel:
meuChevette.ano = 2005;
```

A classe *Chevette* é definida a partir da classe *Automóvel* utilizando herança. A classe *Automóvel* pode possuir outras herdeiras?



DelRey.java

```
class DelRey
    extends Automóvel {

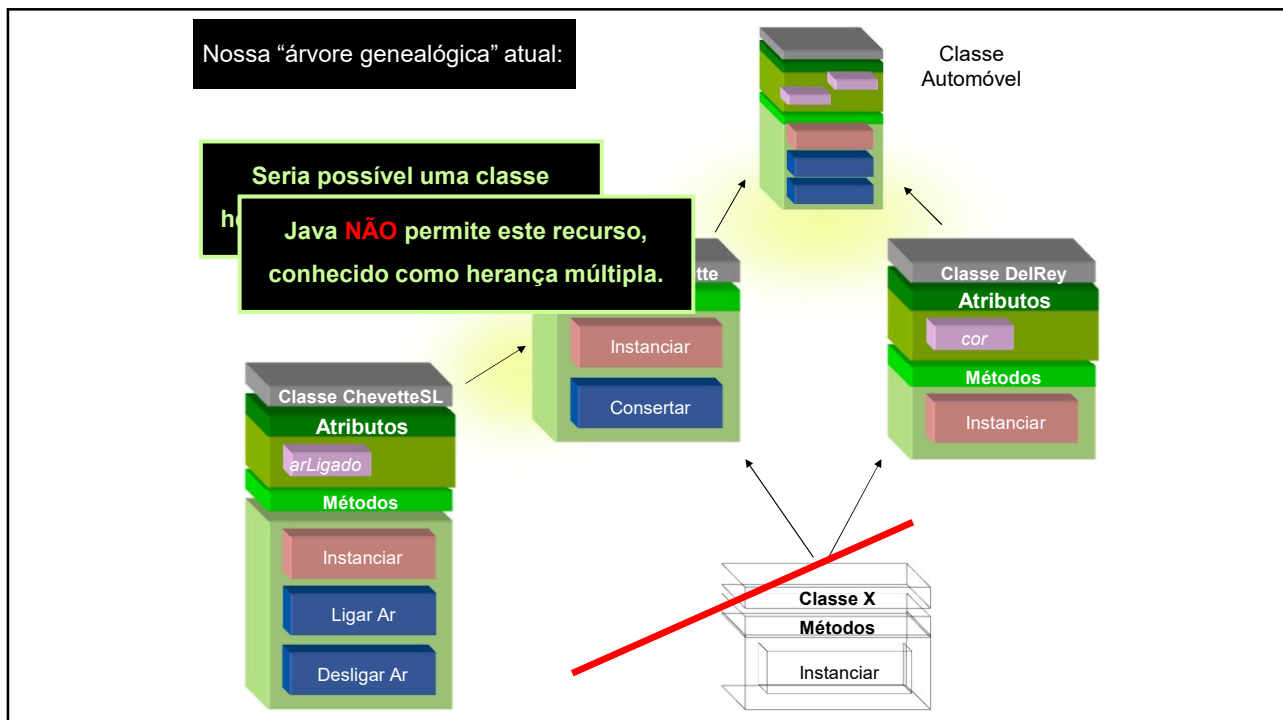
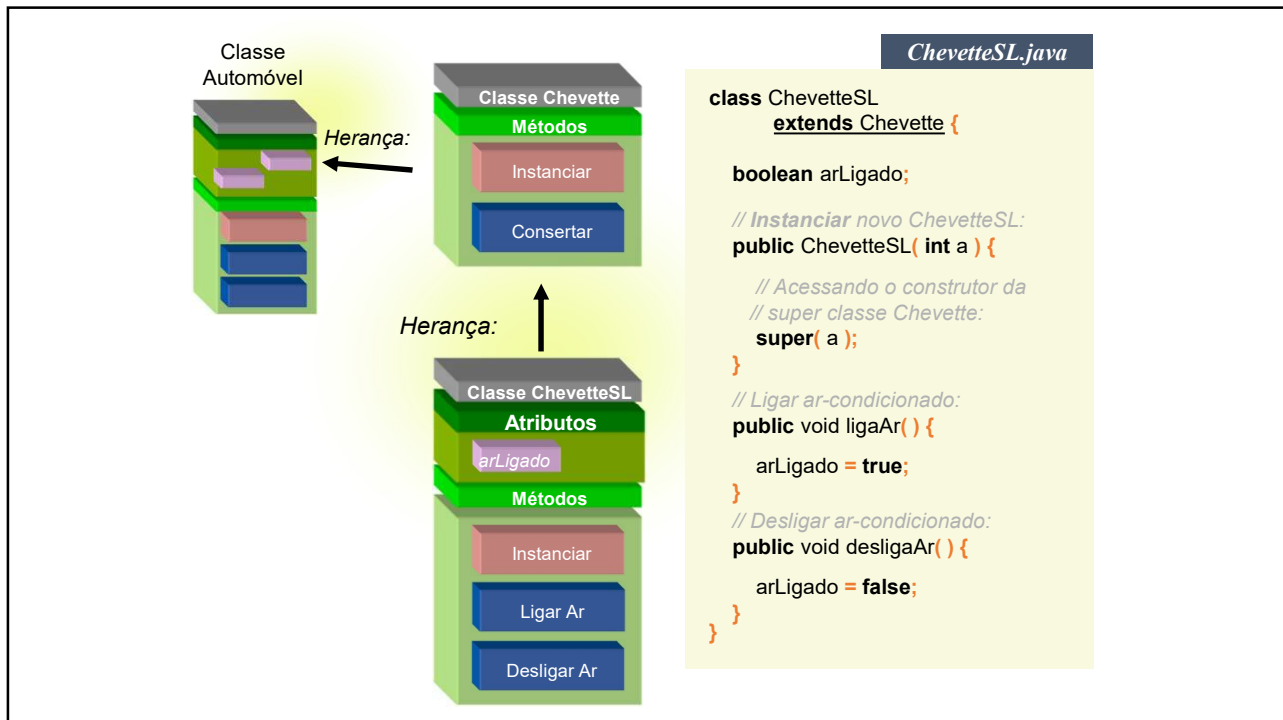
    private String cor;

    // Instanciar novo DelRey:
    public DelRey( int a, String c ) {

        // Acessando o construtor da
        // classe Automóvel:
        super( a );

        cor = c;
    }
}
```

A classe *Chevette* foi definida a partir da classe *Automóvel* utilizando herança. Podemos criar agora classes herdeiras de *Chevette*?



instanceOf (is a)

- ☐ Crie instâncias de todas as classes ...
- ☐ Verifique o *instanceOf* de todas as instâncias contra todas as classes
- ☐ Explique

//Rascunho

```
ChevetteSL csl = new ChevetteSL (1990);

System.out.println(csl instanceof ChevetteSL); //true
System.out.println(csl instanceof Chevette); //true
System.out.println(csl instanceof DelRey); //false
System.out.println(csl instanceof Automovel); //true
System.out.println(csl instanceof Object); //true
```

Olha o polimorfismo aí!!!
Veja quantos instanceof == true

Casting

- ☐ Forçando uma classe a “voltar” a ser enxergada por uma das suas interfaces/visões ou classe de criação (runtime class)

```
public MountainBike myBike = new MountainBike(); // myBike is of type MountainBike.
```

- ☐ MountainBike is descended from Bicycle and Object. Therefore, a MountainBike is a Bicycle and is also an Object, and it can be used wherever Bicycle or Object objects are called for.
- ☐ The reverse is not necessarily true: a Bicycle may be a MountainBike or not. Similarly, an Object may be a Bicycle or a MountainBike, but it isn't necessarily.
- ☐ Casting shows the use of an object of one type in place of another type, **among the objects permitted by inheritance and implementations.**

Casting e instanceof

```
Object obj = new MountainBike(); // ok!!
```

- ❑ then obj is both an Object and a MountainBike (until such time as obj is assigned another object that is not a MountainBike). This is called implicit casting.
- ❑ If, on the other hand, we write

```
MountainBike myBike = obj; // Erro!!!
```

- ❑ Compile-time error because obj is not known to the compiler to be a MountainBike. However, we can tell the compiler that we promise to assign a MountainBike to obj by explicit casting:

```
MountainBike myBike = (MountainBike) obj; // obj volta a ser visto como ...
```

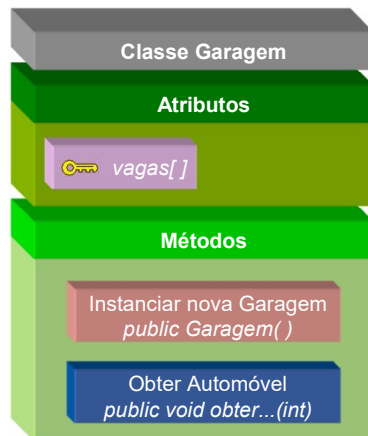
- ❑ This cast inserts a runtime check that obj is assigned a MountainBike so that the compiler can safely assume that obj is a MountainBike. **If obj is not a MountainBike at runtime, an exception will be thrown.**

Verificando classe de runtime com instanceof

- ❑ Note: You can make a logical test as to the type of a particular object using the instanceof operator. This can save you from a runtime error owing to an improper cast. For example:

```
if (obj instanceof MountainBike) {
    MountainBike myBike = (MountainBike) obj;
}
```

- ❑ Here the instanceof operator verifies that obj refers to a MountainBike so that we can make the cast with knowledge that there will be no runtime exception thrown.

**Garagem.java**

```
class Garagem {
    private Automóvel vagas[ ];
    // Instanciar nova garagem
    public Garagem() {

        vagas = new Automóvel [ 5 ];

        vagas[ 0 ] = new Automóvel( 76 );
        vagas[ 1 ] = new Chevette( 88 );
        vagas[ 2 ] = new Chevette( 93 );
        vagas[ 3 ] = new ChevetteSL( 88 );
        vagas[ 4 ] = new DelRey( 88, "Branco" );
    }

    // Obter automóvel
    public Automóvel obterAutomóvel( int posição ) {
        return vagas[ posição ];
    }
}
```

Garagem.java

```
class Garagem {
    private Automóvel vagas[ ];
    // Instanciar nova garagem
    public Garagem() {

        vagas = new Automóvel [ 5 ];

        vagas[ 0 ] = new Automóvel( 76 );
        vagas[ 1 ] = new Chevette( 88 );
        vagas[ 2 ] = new Chevette( 93 );
        vagas[ 3 ] = new ChevetteSL( 88 );
        vagas[ 4 ] = new DelRey( 88, "Branco" );
    }

    // Obter automóvel
    public Automóvel obterAutomóvel( int posição ) {
        return vagas[ posição ];
    }
}
```

MeuPrograma2.java

```
class MeuPrograma2 {
    private Garagem minhaGaragem;
    private Automóvel automóvel;

    // Entry point do programa:
    public static void main( String args[ ] ) {

        // Instanciando um novo objeto Garagem::
        minhaGaragem = new Garagem();

        // Imprimindo a classe de cada automóvel:
        for ( int i = 0; i < 5; i++ ) {

            automóvel = minhaGaragem.obterAutomóvel( i );
            System.out.println( automóvel.getClass().getName() );
        }
    }
}
```

MeuPrograma2.java

```

class MeuPrograma2 {
    private Garagem minhaGaragem;
    private Automóvel automóvel;

    // Entry point do programa
    public static void main( String args[] ) {

        // Instanciando um novo objeto Garagem::
        minhaGaragem = new Garagem();

        // Imprimindo a classe de cada automóvel:
        for ( int i = 0; i < 5; i++ ) {

            automóvel = minhaGaragem.obterAutomóvel( i );
            System.out.println( automóvel.getClass().getName() );
        }
        (continuando...)
    }
}

```

Garagem.java

```

class Garagem {
    private Automóvel[] vagas = new Automóvel[ 5 ];

    public Automóvel obterAutomóvel( int posição ) {
        return vagas[ posição ];
    }
}

```

```

C:\>javac *.java
C:\>dir
Automóvel.class
Chevette.class
ChevetteSL.class
DelRey.class
Garagem.class
MeuPrograma2.class

C:\>java MeuPrograma2
Automóvel
Chevette
Chevette
ChevetteSL
DelRey
C:\>_

```

MeuPrograma2.java

```

class MeuPrograma2 {
    private Garagem minhaGaragem;
    private Automóvel automóvel;

    // Entry point do programa:
    public static void main( String args[] ) {

        // Instanciando um novo objeto Garagem::
        minhaGaragem = new Garagem();

        // Imprimindo a classe de cada automóvel:
        for ( int i = 0; i < 5; i++ ) {

            automóvel = minhaGaragem.obterAutomóvel( i );
            System.out.println( automóvel.getClass().getName() );
        }
        (continuando...)
    }
}

```

MeuPrograma2.java

```
// Imprimindo a classe de cada automóvel:
for ( int i = 0; i < 5; i++ ){

    automóvel = minhaGaragem.obterAutomóvel( i );
    System.out.println( automóvel.getClass().getName() );
}
(continuando...)

// Sabemos que a posição 3 possui um ChevetteSL.
// Vamos tentar obter esse carro e ligar o ar condicionado:
automóvel = minhaGaragem.obterAutomóvel( 3 );
automóvel.ligaAr( true );

// O código correto seria (complete)
= (ChevetteSL) automóvel;
.ligaAr( true );
}
}
```

ERRO DE COMPILAÇÃO!!

Embora a “runtime class” da variável *automóvel* seja *ChevetteSL*, essa variável foi declarada como sendo *Automóvel*, e a classe *Automóvel* não possui o método “*ligaAr(...)*”