

PG II

Programação Orientada por Objectos em Java

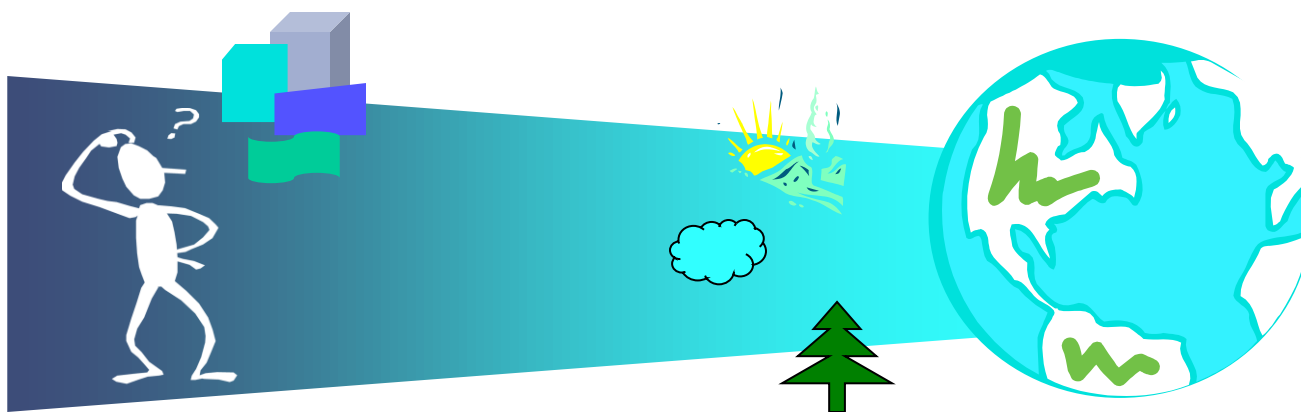
Objectos e Classes:

- Como surgiu OOP?
- Exemplo do Carro
- Classe
- Instâncias
- Ambiente de Classe e de Instância
- Construtor
- OOP
- Conclusões

Como surgiu a Programação Orientadas por Objectos?

Os conceitos fundamentais da **programação orientada por objectos** têm origem na filosofia grega, Sócrates e Platão, nos séculos V e VI AC, sobre:

- a forma ideal de realizar a catalogação ou classificação do conhecimento.

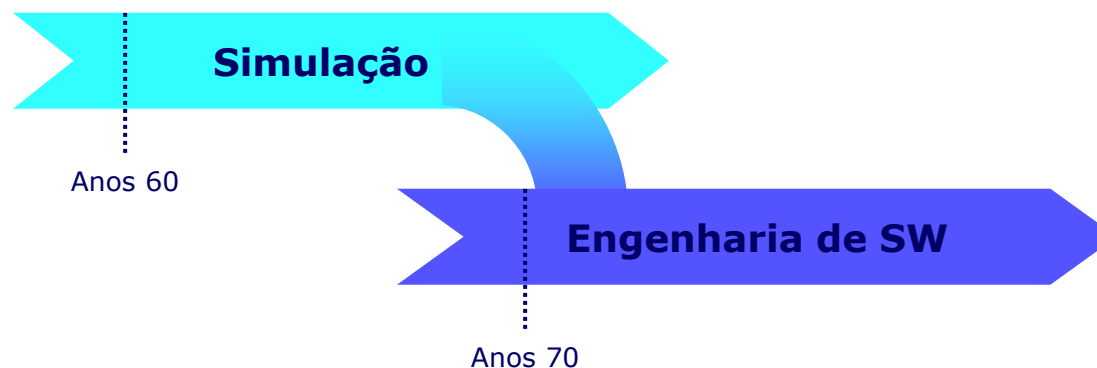


... Como surgiu a Programação Orientadas por Objectos?

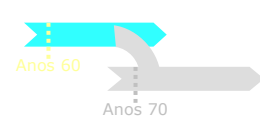
Os primeiros desenvolvimentos, que serviram de base às tecnologias de objectos, usadas hoje em praticamente todas as áreas de informática, surgiram em duas áreas de conhecimento bem distintas:

- **Simulação;**
- **Engenharia de *Software*.**

Da intercepção de **conceitos**, que evoluíram na área da Simulação e da Engenharia de Software aparece a definição da **Programação Orientada por Objectos**:



Principiais conceitos provenientes da **Simulação**



Os principais conceitos provenientes da área de conhecimento da **Simulação**, surgiram no final dos **anos 60**, na linguagem de programação **SIMULA67**.

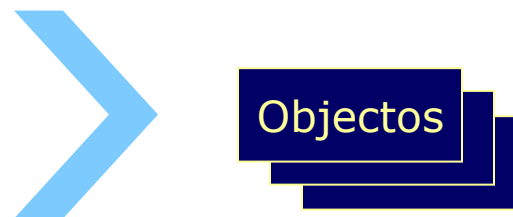
Esta linguagem tinha como fim o desenvolvimento de **modelos** do mundo real e sobre estes a execução de simulações.

Objectivo:

- Encontrar formas simples de representar (modelizar) em computador entidades do mundo real, cujo comportamento se pretende analisar, sendo reconhecido que tais entidades possuem atributos definitórios próprios, isto é, valores associados a propriedades (variáveis) que representam a sua estrutura interna.

Desta abordagem concluiu-se que as **entidade do mundo real** são **modeláveis** através das seguintes informações:

- **Estrutura** (definida por atributos);
- **Comportamento** (acções <> reacções);
- **Interacção** (relação com outras entidades);
- **Identidade** (única).



... Principiais conceitos provenientes da **Simulação**



Com base nos conceitos da Simulação, um **possível modelo** para a representação de um veículo automóvel será:

- **Estrutura** (atributos): cilindrada (cm³*) e potência (cv's);
- **Comportamento** (acções <> reacções): velocidade, acelerar, etc;
- **Interacção**: sinais de luzes e buzina;
- **Identidade**: única.



Objecto A

- **Estrutura**: 1.500 cm³ e 90 cv
- **Comportamento**:
 - Curvas: 90 Km/h;
 - Auto-estrada: 180 Km/h;
 - Fora de estrada: 20 Km/h;
 - Etc...

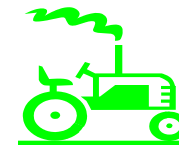
» **Identidade: carro**



Objecto B

- **Estrutura**: 6.000 cm³ e 150 cv
- **Comportamento**:
 - Curvas: 40 Km/h;
 - Auto-estrada: 100 Km/h;
 - Fora de estrada: 30 Km/h;
 - Etc...

» **Identidade: camião**



Objecto C

- **Estrutura**: 3.500 cm³ e 80 cv
- **Comportamento**:
 - Curvas: 40 Km/h;
 - Auto-estrada: 40 Km/h;
 - Fora de estrada: 40 Km/h;
 - Etc...

» **Identidade: tractor**

(*) Ao definir que a cilindrada se mede em cm³ está-se a delimitar o universo (conjunto) de valores que a variável cilindrada poderá assumir. Ou seja, "cm³" é o tipo da variável cilindrada e "cv's" é o tipo da variável potência.

... Principiais conceitos provenientes da **Simulação**



O exemplo anterior mostra uma forma de **modelar** um veículo automóvel cumprindo os conceitos provenientes da Simulação:

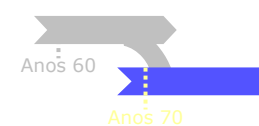
- Os Objectos A, B e C têm identidades e propriedades diferentes, mas apresentam características (variáveis) iguais.
- Os Objectos A, B e C obedecem a um mesmo padrão comum, de estrutura e comportamento.
- Os Objectos A, B, e C são definidos por uma:
 - Estrutura: cilindrada (cm³*) e potência (cv's);
 - Comportamento: velocidade.
- Denomina-se a entidade geradora e agrupadora dos objectos A, B e C de **Classe**.
- A **Classe** dos Objectos A, B e C a que correspondem respectivamente o Carro, o Camião e o Tractor é definida por:

- **Estrutura**: matrícula, cilindrada (cm³*) e potência (cv's);
- **Comportamento**: velocidade.



Classe

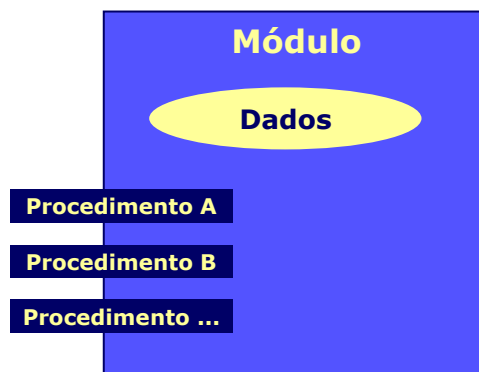
A evolução da Engenharia de Software



Num determinado momento da evolução da Engenharia de Software, a programação tinha algumas dificuldades, entre as quais:

- A necessidade de criar objectos (além dos tipos primitivos) para representar entidades;
- A reutilização de **tipos** tinha pouca utilidade, sem a existência de funções próprias.

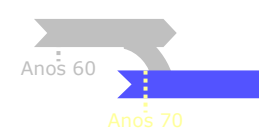
Da conjugação destas necessidades com os conceitos introduzidos pela Simulação, surgem as unidades de computação independentes, designadas por **Módulos**:



Os módulos apresentam as seguintes características:

- Compiláveis separadamente;
- Contêm declarações de: **Dados** + **Procedimentos**.

Exemplo do Carro



Aplicando o conceito de **Módulo**, o objecto A (Carro) pode ser definido em Java da seguinte forma:

```
Carro
1 package aula02;
2
3 public class Carro {
4     private static int cilindrada;
5     private static int potencia;
6     private static int mudanca;
7     private static int MAX;
8
9     public static double velocidadeMax() {
10         double vel = ((cilindrada/40)+((potencia/8)/mudanca))*mudanca;
11         if (vel > MAX) vel = MAX;
12         return vel;
13     }
14
15     public static void main (String args[]) {
16         cilindrada = 1500;
17         potencia = 120;
18         MAX=200;
19
20         pg2.io.IO.cout.writeln ("Mudanca: ");
21         mudanca = pg2.io.IO.cin.readInt();
22         pg2.io.IO.cout.writeln ("0 carro em " + mudanca +
23                                 "a tem a velocidade maxima de: " +
24                                 velocidadeMax() + "kms/h");
25     }
26 }
```

Teste:

```
Command Prompt
D:\work>java aula02.Carro
Mudanca:
1
0 carro em 1a tem a velocidade maxima de: 52.0kms/h
D:\work>java aula02.Carro
Mudanca:
2
0 carro em 2a tem a velocidade maxima de: 88.0kms/h
D:\work>java aula02.Carro
Mudanca:
3
0 carro em 3a tem a velocidade maxima de: 126.0kms/h
D:\work>java aula02.Carro
Mudanca:
4
0 carro em 4a tem a velocidade maxima de: 160.0kms/h
D:\work>java aula02.Carro
Mudanca:
5
0 carro em 5a tem a velocidade maxima de: 200.0kms/h
D:\work>_
```


... Exemplo do Carro

O exemplo anterior cumpre os requisitos de **Módulo**:

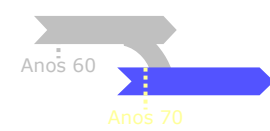
- Unidade de programação compilável - Classe → `"javac aula02.Carro";`
- Contem declarações de: **Dados + Procedimentos**:



Os princípios da programação dos módulos assentam então nas seguintes regras:

- **Estrutura** definida por um conjunto de **variáveis** globais;
- Conjunto específico de funções, para manipulação do módulo a partir do exterior, designado de **interface** (proveniente da noção inglesa: API – *Application Programming Interface*);
- Os módulos são **cápsulas** que escondem os detalhes de programação ao exterior e disponibilizam um conjunto de procedimentos específicos para a sua manipulação, criando uma abstracção à sua estrutura de dados interna.

... E se quisermos implementar o objecto Tractor ?



Como garantir que objectos iguais têm a mesma estrutura interna e comportamento?

- Mesmas **variáveis** (atributos);
- Mesmas **funções**.

1ª Hipótese: “Copy & paste” do código do objecto e atribuição de novos valores à estrutura.

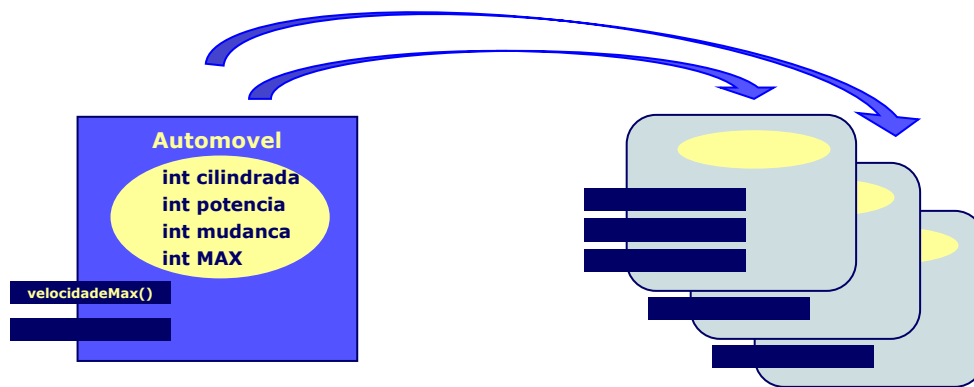
```
Tractor
1 package aula02;
2
3 public class Tractor {
4     private static int cilindrada;
5     private static int potencia;
6     private static int mudanca;
7     private static int MAX;
8
9     public static double velocidadeMax() {
10         double vel = ((cilindrada/40)+((potencia/8)/mudanca))*mudanca;
11         if (vel > MAX) vel = MAX;
12         return vel;
13     }
14
15     public static void main (String args[]) {
16         cilindrada = 3500;
17         potencia = 80;
18         MAX = 40;
19
20         ...
21     }
22 }
```

2ª Hipótese:

Criar a partir de uma **única classe**, diversos objectos com:

- a estrutura interna (variáveis);
- e a implementação (funções),

que pretendemos ver definida em todos os objectos daquele tipo.



```
Automovel
1 package aula02;
2
3 public class Automovel {
4     private static int cilindrada;
5     private static int potencia;
6     private static int mudanca;
7     private static int MAX;
8
9     public static double velocidadeMax() {
10         double vel = ((cilindrada/40)+((potencia/8)*mudanca))*mudanca;
11         if (vel > MAX) vel = MAX;
12         return vel;
13     }
14 }
```

A **Classe** dos Objectos A, B e C a que correspondem respectivamente o Carro, o Camião e o Tractor, será designada por **Automovel** e é definida por:

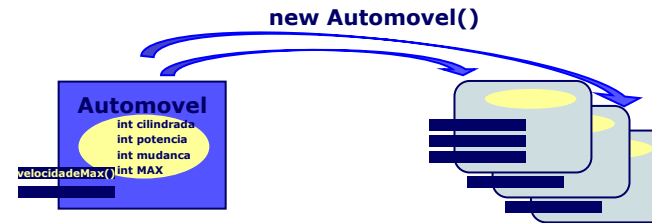
- **Estrutura:** cilindrada, potencia, mudanca e MAX;
- **Comportamento ou interface:** velocidadeMax().



Classe

Para que a partir da classe Automovel seja criado o objecto Carro, Camião ou Tractor, bastará usar a instrução **new** seguido do nome da respectiva classe e "()":

```
...  
new Automovel();  
new Automovel();  
new Automovel();  
...
```

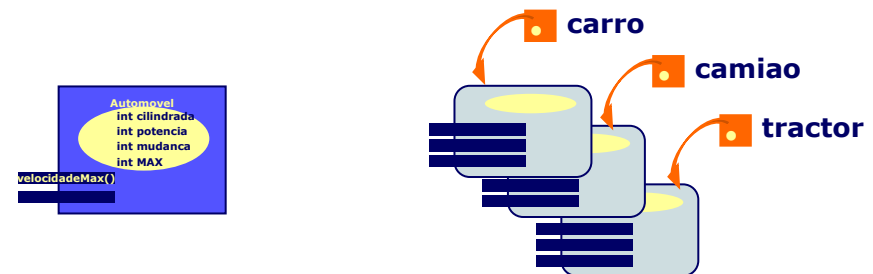


Estas 3 instruções poderiam ser executadas, por exemplo na função main() da classe Automovel, criando 3 objectos, ou **instâncias**, que seriam de seguida destruídos. **Porquê?**

Porque, não existe nenhuma referência para estes objectos, ou seja, não existe nenhuma **variável** que "aponte" para eles.

A forma de guardar as referência para esses Objectos, é através de variáveis do mesmo **tipo** (Automovel), que receberão o retorno da instrução "**new Automovel()**":

```
...  
Automovel carro = new Automovel();  
Automovel camiao = new Automovel();  
Automovel tractor = new Automovel();  
...
```



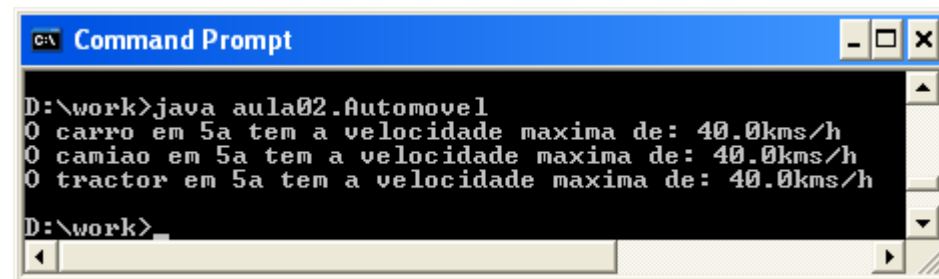
... Instâncias

Através das referências para os objectos podemos aceder à sua **estrutura** e **interface**:

```

14  ...
15  public static void main (String args[]) {
16      Automovel carro = new Automovel();
17      Automovel camiao = new Automovel();
18      Automovel tractor = new Automovel();
19
20      carro.cilindrada = 1500;
21      carro.potencia = 120;
22      carro.mudanca = 5;
23      carro.MAX = 200;
24      camiao.cilindrada = 6000;
25      camiao.potencia = 150;
26      camiao.mudanca = 5;
27      camiao.MAX = 120;
28      tractor.cilindrada = 3500;
29      tractor.potencia = 80;
30      tractor.mudanca = 5;
31      tractor.MAX = 40;
32      pg2.io.IO.cout.writeln ("O carro em " + carro.mudanca +
33                              "a tem a velocidade maxima de: " +
34                              carro.velocidadeMax() + "kms/h");
35      pg2.io.IO.cout.writeln ("O camiao em " + camiao.mudanca +
36                              "a tem a velocidade maxima de: " +
37                              camiao.velocidadeMax() + "kms/h");
38      pg2.io.IO.cout.writeln ("O tractor em " + tractor.mudanca +
39                              "a tem a velocidade maxima de: " +
40                              tractor.velocidadeMax() + "kms/h");
41
42  }
43  }
    
```

Contudo ao executarmos esta classe o resultado pode parecer curioso:

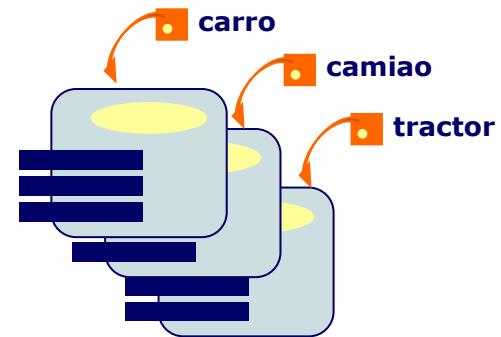


```

D:\work>java aula02.Automovel
O carro em 5a tem a velocidade maxima de: 40.0kms/h
O camiao em 5a tem a velocidade maxima de: 40.0kms/h
O tractor em 5a tem a velocidade maxima de: 40.0kms/h
D:\work>
    
```

Porquê a mesma velocidade para os diferentes objectos?

Nenhuma incorrecção foi cometida em nenhum dos passos exemplificados. Até mesmo, o esquema apresentado pode ser considerado correcto e da sua análise poder-se-á entender o porquê do resultado anterior:



Os três objectos referenciados pelas variáveis, *carro*, *camiao* e *tractor*, foram criados a partir da classe Automovel, que é **única**. Todas as **variáveis** e **funções** definidas como “**static**” pertencem ao ambiente desta classe e também são **únicas**.

Neste contexto, as instruções seguintes são equivalentes, ou seja, manipulam a mesma variável:

```
carro.cilindrada = 1500;  
camiao.cilindrada = 6000;  
tractor.cilindrada = 3500;  
Automovel.cilindrada = 9999;
```

A variável cilindrada é uma variável da classe Automóvel e ficará com o valor da última instrução, isto é, **9999**.

... Ambiente de Classe e de Instância

Para que cada uma das instâncias possa manipular um conjunto de **variáveis** “seu” e independente das outras instâncias, então deverá ser omitido o prefixo “**static**” da declaração de cada variável.

Por sua vez, a função **velocidadeMax()**, que com base no valor destas variáveis calcula a respectiva velocidade máxima também deverá deixar de ser “static”.

De notar, que este **método** (velocidadeMax) retorna um resultado em função do **estado interno do objecto** (cilindrada, mudança, etc), que é implementado pelas suas **variáveis de instância**.

A definição da classe **Automovel** passa a estar assim de acordo com a seguinte implementação:

```

Automovel
1 package aula02;
2
3 public class Automovel {
4     private int cilindrada;
5     private int potencia;
6     private int mudanca;
7     private int MAX;
8
9     public double velocidadeMax() {
10         double vel = ((cilindrada/40)+((potencia/8)/mudanca))*mudanca;
11         if (vel > MAX) vel = MAX;
12         return vel;
13     }
14

```

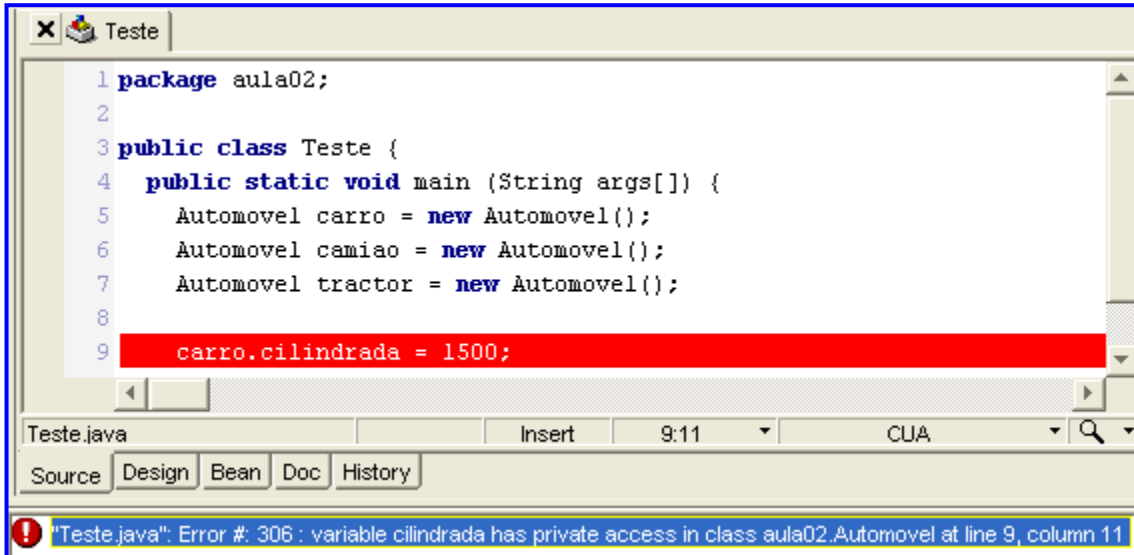
Repetindo o mesmo teste que foi executado anteriormente seria agora obtido o seguinte resultado:

```

Command Prompt
D:\work>java aula02.Automovel
0 carro em 5a tem a velocidade maxima de: 200.0kms/h
0 camiao em 5a tem a velocidade maxima de: 120.0kms/h
0 tractor em 5a tem a velocidade maxima de: 40.0kms/h

```

Se numa outra classe qualquer (classe Teste) for criada uma instancia de Automovel, não existe forma de afectar a sua estrutura (variáveis de instância):



```
1 package aula02;
2
3 public class Teste {
4     public static void main (String args[]) {
5         Automovel carro = new Automovel();
6         Automovel camiao = new Automovel();
7         Automovel tractor = new Automovel();
8
9         carro.cilindrada = 1500;
```

De imediato o compilador reclama que a variável *cilindrada* é *private*, pelo que não poderá ser acedida, excepto dentro da classe *Automovel*.

Algumas das soluções para este problema seriam:

- Dar acesso público às respectivas variáveis. Contudo, não sendo este o caso, podem existir situações em que não seja conveniente dar acesso directo à **estrutura** de um objecto.
- Adicionar à classe *Automovel* **métodos** específicos para este efeito, acessíveis do "exterior":

```
public void setCilindrada(int c){cilindrada = c;}
public void setPotencia(int p){potencia = p;}
...
```

Existem várias soluções para um problema, cada uma com as suas vantagens e desvantagens.

... Construtor

E não existiria uma solução mais expedita, que no momento da criação do objecto permitisse a atribuição imediata dos valores da estrutura?

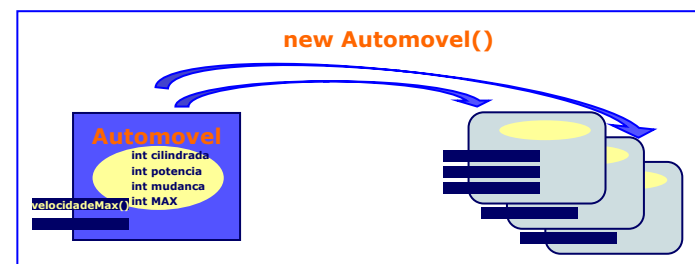
Algo como: `Automovel carro = new Carro(1500, 120, 5, 200)`, onde cada um dos valores passados como parâmetros eram atribuídos à *cilindrada*, *potencia*, *mudanca* e *MAX*, respectivamente.

Sim, é possível. Para isso deve ser introduzido o conceito de **construtor**.

O construtor de uma classe:

- Tem como função a criação de objectos dessa classe;
- É usado com a instrução **new**;
- É um método especial cuja designação é igual ao nome da classe.

Por omissão qualquer classe que não tenha **explicitamente** definido um construtor, tem **implícito** um construtor sem parâmetros.



No caso da classe Automovel não foi definido qualquer construtor, no entanto é possível implementar a instrução: `Automovel carro = new Automovel()`

O construtor Automovel() está implícito na classe.

... Construtor

Para que seja então possível fazer: `Automovel carro = new Carro(1500, 120, 5, 200)`
teremos que definir o respectivo construtor na classe Automovel:

```

9  public Automovel(int c, int p, int m, int max){
10      cilindrada = c;
11      potencia = p;
12      mudanca = m;
13      MAX = max;
14  }

```

De notar, que a forma de declarar um construtor é semelhante à de um método, mas que não tem declarado o tipo de retorno.

Assim, a classe Teste podia ser concluída da seguinte forma:

```

1 package aula02;
2 public class Teste {
3     public static void main (String args[]) {
4         Automovel carro = new Automovel(1500, 120, 5, 200);
5         Automovel camiao = new Automovel(6000, 150, 5, 120);
6         Automovel tractor = new Automovel(3500, 80, 5, 40);
7         pg2.io.IO.cout.writeln ("0 carro tem a velocidade maxima de: " +
8                                 carro.velocidadeMax() + "kms/h");
9         pg2.io.IO.cout.writeln ("0 camiao tem a velocidade maxima de: " +
10                                camiao.velocidadeMax() + "kms/h");
11         pg2.io.IO.cout.writeln ("0 tractor tem a velocidade maxima de: " +
12                                 tractor.velocidadeMax() + "kms/h");
13     }
14 }

```

```

D:\work>java aula02.Teste
0 carro tem a velocidade maxima de: 200.0kms/h
0 camiao tem a velocidade maxima de: 120.0kms/h
0 tractor tem a velocidade maxima de: 40.0kms/h
D:\work>

```

Para concluir a definição do modelo do Automovel, faz sentido que:

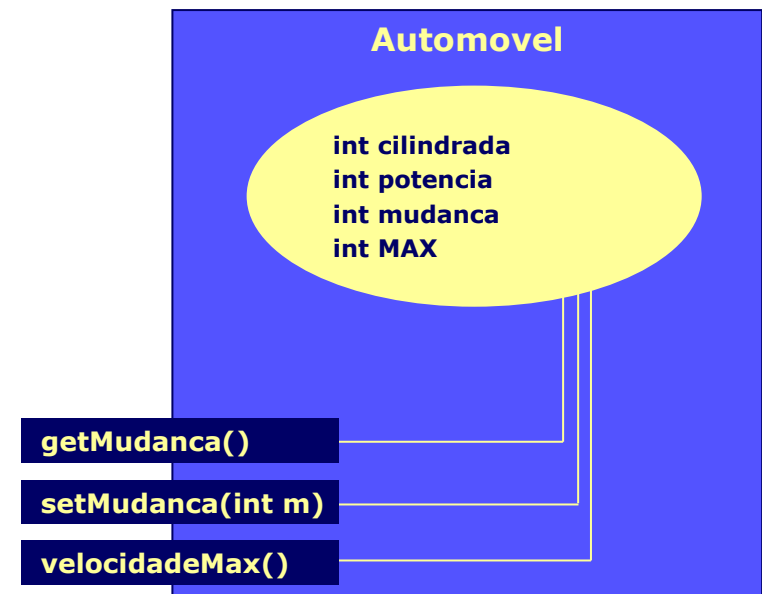
- Existam métodos específicos para consultar e afectar a variável *mudanca*;
- As variáveis *cilindrada*, *potencia* e *MAX* continuem escondidas do exterior. Ou seja, uma vez instanciado um Automovel, estes atributos não devem ser alterados;
- Quando um Automovel for instanciado a *mudanca* deve ficar com o valor zero, o que pode ser interpretado como estando em ponto morto.

Nestes pressupostos a definição final da classe Automovel poderia ser:

```

1 package aula02;
2
3 public class Automovel {
4     private int cilindrada;
5     private int potencia;
6     private int mudanca;
7     private int MAX;
8
9     public Automovel(int c, int p, int max){
10         cilindrada = c;
11         potencia = p;
12         MAX = max;
13     }
14     public int getMudanca(){return mudanca;}
15     public void setMudanca(int m){mudanca=m;}
16     public double velocidadeMax() {
17         double vel = ((cilindrada/40)+((potencia/8)/mudanca))*mudanca;
18         if (vel > MAX) vel = MAX;
19         return vel;
20     }

```



Mantendo a mesma interface, uma outra forma de ver a implementação da classe Automovel era no pressuposto que:

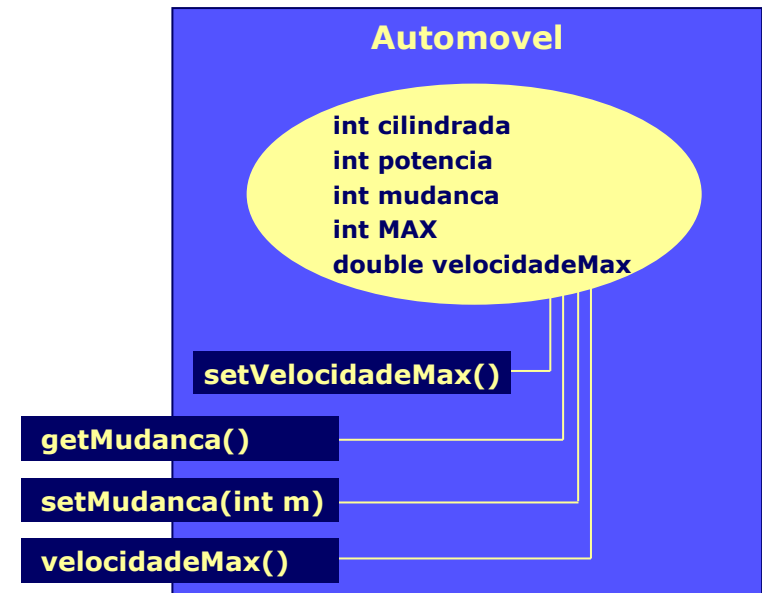
- Sempre que há uma alteração da mudança do veículo (*setMudanca*), deve ser calculado de imediato a velocidade máxima que este poderá atingir;
- O método *velocidadeMax* deverá ser apenas um método de consulta de um atributo do Automovel, que já foi actualizado no momento de afectação da mudança (*setMudanca*).

Nestes novos pressupostos uma outra implementação para a classe Automovel poderia ser:

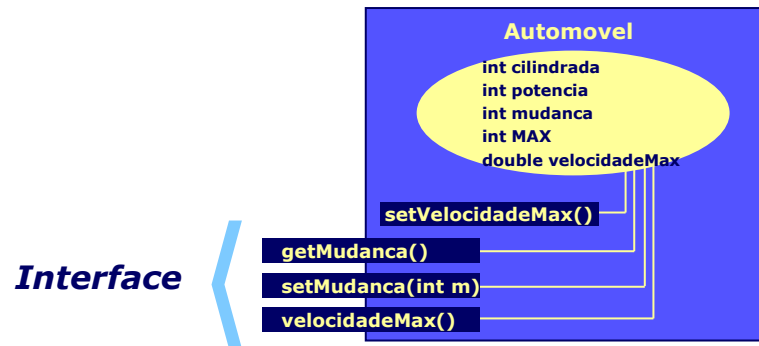
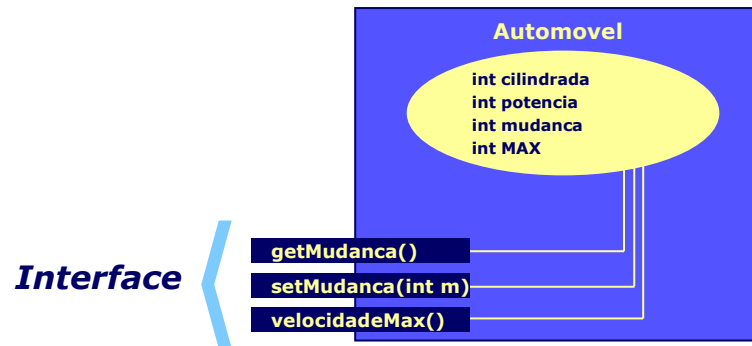
```

1 package aula02;
2
3 public class Automovel {
4     private int cilindrada;
5     private int potencia;
6     private int mudanca;
7     private int MAX;
8     private double velocidadeMax;
9     public Automovel(int c, int p, int max){
10         cilindrada = c; potencia = p; MAX = max;
11     }
12     public int getMudanca(){return mudanca;}
13     public void setMudanca(int m){
14         mudanca = m;
15         setVelocidadeMax();
16     }
17     private void setVelocidadeMax(){
18         velocidadeMax = ((cilindrada/40)+((potencia/8)/mudanca))*mudanca;
19         if (velocidadeMax > MAX) velocidadeMax = MAX;
20     }
21     public double velocidadeMax(){return velocidadeMax;}

```



Estas duas possíveis implementações da classe Automovel, apresentam diferenças ao nível interno (daquilo que é *private*), mas mantêm a mesma interface (o que foi definido como *public*)

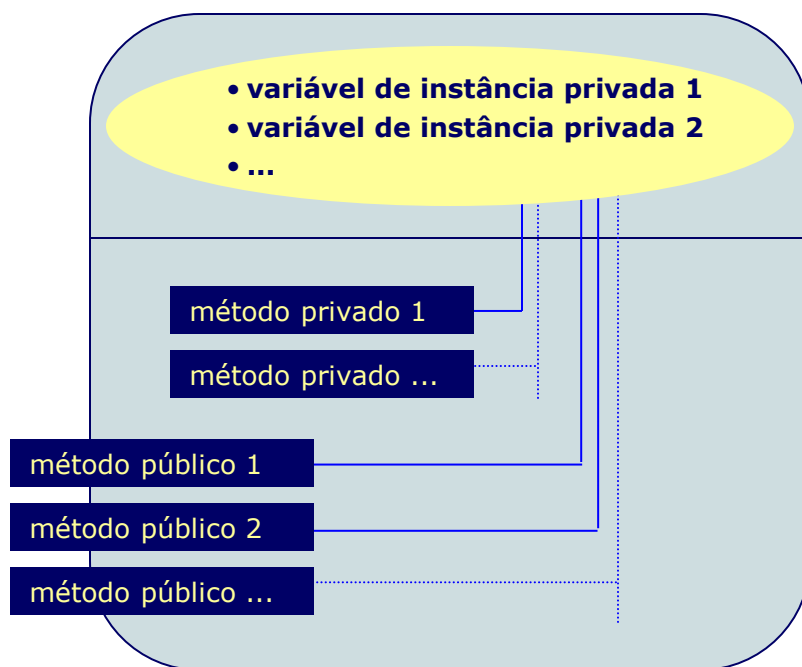


Ou seja, qualquer classe de Teste externa, poderia reutilizar qualquer umas das implementações da classe Automovel sem que tivesse que alterar qualquer linha de código.

Conclusões:

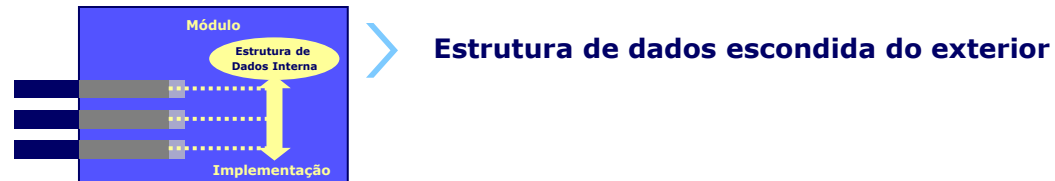
- Uma Classe pode ter diferentes implementações, conservando o mesmo comportamento;
- Para manter a compatibilidade do código desenvolvido deve ser mantida a abstracção da **implementação** e da **estrutura de dados** do objecto usado.

Com base nos conceitos anteriores passamos a ter uma nova representação para os objectos de acordo com a figura seguinte:



Algumas regras no contexto da OOP são:

- Encapsulamento:



- Abstracção:

Manipulação através do interface específico



- Modularidade e Reutilização:

