Formabase
Aprenda passo a passo

Java
Programação Orientada por Objetos

Metodologia

- Interprete o documento calmamente e com atenção.
- Acompanhe a execução do exercício no seu computador.
- Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

Conteúdo programático

1. Introdução

Até aqui utilizámos o Java para perceber a lógica e semântica gerais de programação, fazendo uso apenas de uma classe e um método principal `main`. Ignorámos assim, por questões de simplificação, a ideia base desta linguagem de programação orientada por objetos. Este é um modelo de programação mais organizado que utiliza o conceito de objetos, em vez de uma longa lista de instruções e métodos como temos vindo a observar.

Um objeto não é mais do que uma estrutura de código que representa uma determinada entidade, como uma pessoa, um carro, uma conta bancária, etc. Tal como no mundo real, todas as entidades possuem estados, que podem mudar ao longo do tempo, e comportamentos que podem alterar os seus estados ou afetar até outras entidades. A tabela seguinte descreve alguns exemplos:

Objeto	**Estados**	**Comportamentos**
Pessoa	Nome, data de nascimento, C.C., altura	Falar, andar, pensar, comer, conduzir
Carro	Matrícula, marca, velocidade, cor, modelo	Travar, acelerar, virar, estacionar
Conta bancária	Saldo, NIB, taxa de juro	Depositar, creditar, levantar dinheiro

Ao longo de um programa utilizamos frequentemente várias entidades que partilham as mesmas propriedades. Torna-se assim necessário produzir um modelo que permita criar vários objetos do mesmo tipo. A este modelo damos o nome de **classe**, e contém todos os estados (**atributos**) de um determinado tipo de objetos, bem como os seus comportamentos (**métodos**).

Em programação, os estados ou informações de um objeto chamam-se atributos e correspondem a variáveis. Os comportamentos têm o nome de métodos e correspondem a funções.

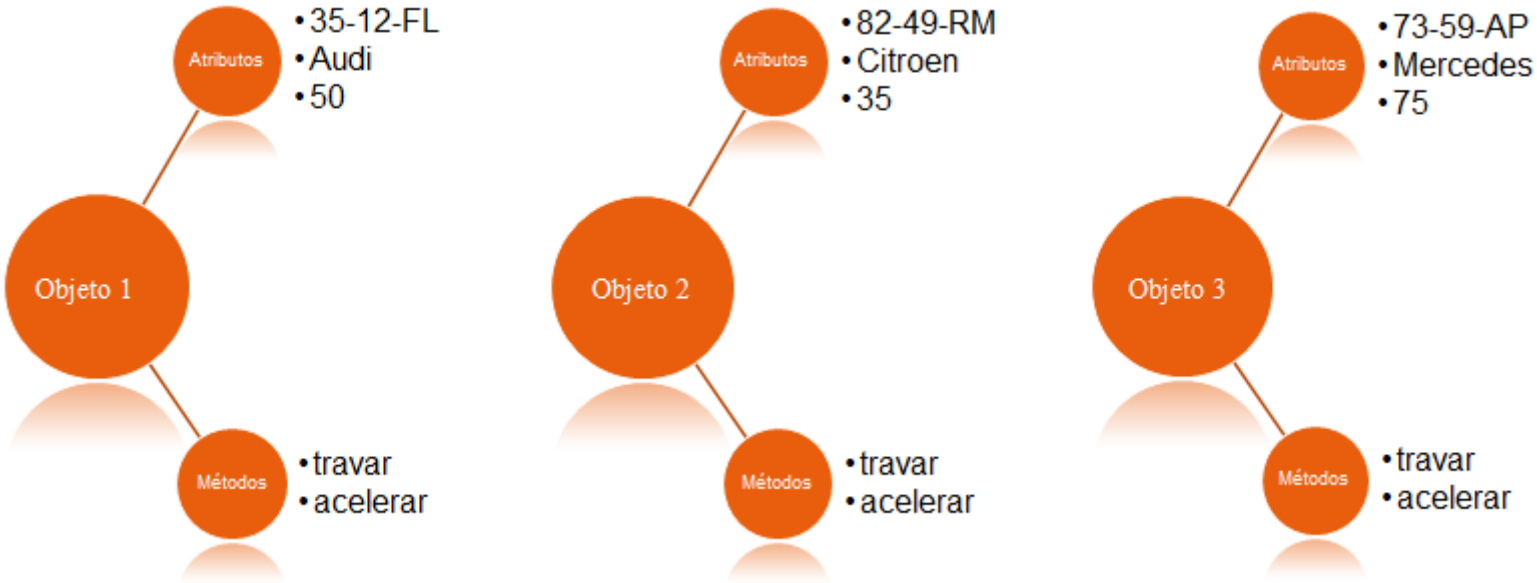
Desta forma, usamos as classes como se se tratassem de moldes, que definem a estrutura geral dos objetos que precisamos ao longo do código. Este modelo de programação traz grandes vantagens a nível de desenvolvimento, organização e manutenção do código dos nossos programas, principalmente para projetos de grande escala onde se trabalha numa equipa.

Imaginemos que pretendíamos utilizar o conceito de carro num programa. Iríamos então ter objetos do tipo “Carro”, que seriam construídos através de uma classe representada no seguinte diagrama:

Através do diagrama conseguimos verificar que a entidade "Carro" possui as ações de "travar" e "acelerar", assim como as informações para "marca", "matrícula" e "velocidade". No entanto, cada objeto criado através deste molde irá ter as suas informações e ações específicas. Significa isto que através desta classe pode ser criado um objeto "Carro" de marca Audi e outro objeto "Carro" de marca Mercedes.

A cada objeto criado através de uma classe damos o nome de instância.

Se criarmos três objetos através desta classe Carro, e atribuirmos diferentes informações a cada um deles, ficamos com três instâncias de "Carro":



2. Implementação

Vamos agora passar estes conceitos para o código Java para percebermos como funciona.

- **Crie** uma **classe** de nome `Carro.java`

  Esta classe vai definir o molde dos objetos.

- **Introduza** o seguinte código:

```
public class Carro {
    public String marca, matricula;
    public int velocidade;

    public void travar(){
        velocidade = 0;
    }

    public void acelerar(){
        velocidade += 10;
    }
}
```

As duas primeiras linhas definem os atributos da classe e os seus respetivos tipos:

```
public String marca, matricula;
public int velocidade;
```

O restante código trata da implementação dos métodos que definem as ações do objeto. O método `acelerar` por exemplo, soma **10** ao valor do atributo `velocidade` do objeto.

Vamos agora construir a classe que contém o método main com o código para testar a classe `Carro`.

- **Crie** a classe `TesteCarro.java`

- **Insira** o seguinte código:

```
public class TesteCarro {
    public static void main(String[] args) {
```

```
        Carro carro1 = new Carro();
        carro1.velocidade = 20;
        System.out.println(carro1.velocidade);
        carro1.acelerar();
        System.out.println(carro1.velocidade);
    }
}
```

- **Compile** e **teste** a classe

  - Neste exemplo criámos um objeto do tipo carro e definimos a sua velocidade com o valor de **20**. De seguida o método `acelerar` aumentou a sua velocidade.

  - A variável carro1 é uma variável do tipo `Carro`.

O operador `new` cria um novo objeto de uma determinada classe especificada à sua frente:

```
Carro carro1 = new Carro();
```

A seguinte instrução guarda o valor **20** no atributo `velocidade`:

```
carro1.velocidade = 20;
```

Ou seja, para alterar os valores dos atributos de um objeto usa-se a seguinte sintaxe:

```
objeto.atributo = valor;
```

- **Redefina** o código do método `main` de acordo com o seguinte código:

```
public static void main(String[] args) {

    Carro carro1 = new Carro();
    carro1.matricula = "35-12-FL";
    carro1.marca = "Audi";
    carro1.velocidade = 50;

    Carro carro2 = new Carro();
    carro2.matricula = "82-49-RM";
    carro2.marca = "Citroen";
    carro2.velocidade = 35;

    Carro carro3 = new Carro();
    carro3.matricula = "73-59-AP";
    carro3.marca = "Mercedes";
    carro3.velocidade = 75;
}
```

- **Teste** novamente

  - Repare como cada objeto tem informações distintas, apesar de terem sido todos criados a partir do mesmo molde.

  - O código atual do método `main` corresponde à ilustração dos objetos no início deste exercício.

Vamos agora definir uma nova ação na classe `Carro` que permita mostrar a informação de um objeto na consola.

- Na classe `Carro` **adicione** este novo método:

```
public void mostrar(){
    System.out.println();
    System.out.println("Marca -> " + marca);
    System.out.println("Matricula -> " + matricula);
    System.out.println("Velocidade -> " + velocidade);
}
```

  - Este método mostra no ecrã os atributos que o objeto possui no momento.

- No **final** do método `main` da classe `TesteCarro` **adicione** as seguintes instruções:

```
carro1.mostrar();
carro2.mostrar();
carro3.mostrar();
```

- **Compile** e **teste** a classe

  - Confirme que vê na consola as informações dos três objetos.

3. Construtor

Em programação orientada por objetos existe um método especial em cada classe chamado **construtor**. Este método é executado sempre que é criado um novo objeto. Por este motivo é normal serem feitas as inicializações dos atributos da classe neste método. O construtor é um método com o

nome igual ao da classe e não possui tipo de retorno.

A sintaxe de um construtor será então:

```
public Classe(){
}
```

De notar que o construtor funciona como um método, logo pode receber parâmetros. Vamos aplicar este conceito à classe `Carro`:

- **Adicione** o seguinte método à classe `Carro`:

```
public Carro(String matricula_carro , String marca_carro, int velocidade_carro){
    marca = marca_carro;
    matricula = matricula_carro;
    velocidade = velocidade_carro;
}
```

- **Altere** o método `main` na classe `TesteCarro` de maneira a ficar igual ao seguinte:

```
public static void main(String[] args) {
    Carro carro1 = new Carro("35-12-FL","Audi",50);
    Carro carro2 = new Carro("82-49-RM","Citroen",35);
    Carro carro3 = new Carro("73-59-AP","Mercedes",75);

    carro1.mostrar();
    carro2.mostrar();
    carro3.mostrar();
}
```

- **Teste** e **confirme** que o resultado é igual ao anterior

Repare como esta escrita é mais compacta. Para cada criação de um objeto do tipo carro estão a ser especificados os valores para os seus atributos. No entanto, os valores destes atributos podem mudar com o decorrer do programa.

A linha seguinte cria um objeto do tipo carro passando três parâmetros diferentes. Como vimos no módulo dos métodos, estes parâmetros vão ser passados ao método construtor desta classe. Os valores são guardados nos atributos do objeto, neste caso do `carro1`.

```
Carro carro1 = new Carro("35-12-FL","Audi",50);
```

4. Strings

Embora este tipo de variáveis já tenha sido abordado nos primeiros módulos, algumas partes foram omitidas na altura. Isto porque o tipo de dados **String** é na verdade uma classe, que já possui uma coleção de métodos para podermos modificar e controlar o texto guardado como atributo.

Vamos ver exemplos destes métodos:

- **Crie** uma nova **classe** de nome `ManipulacaoStrings.java`:

```
import java.util.Scanner;

public class ManipulacaoStrings {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
```

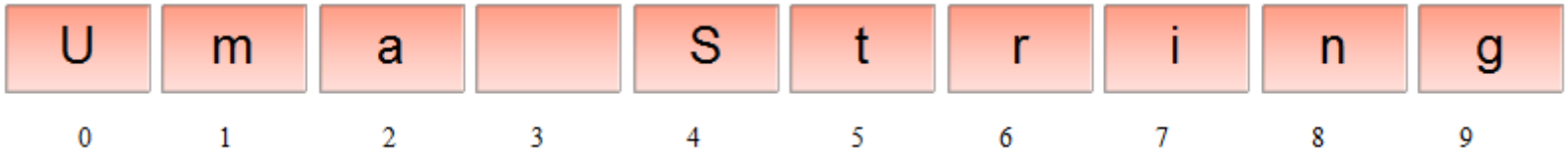
```
        System.out.println("Introduza uma frase");
        String frase = teclado.nextLine();
        String transformada = "";

        for (int i = 0; i < frase.length();++i){
            char corrente = frase.charAt(i);

            if (corrente == 'r'){
                transformada += 'l';
            }
            else {
                transformada += corrente;
            }
        }

        System.out.println("A frase com os r's substituidos por l's : " + transformada);
    }
}
```

- **Compile** e **teste** a classe

Repare como agora foi usado o método `nextLine` para obtermos uma **String** do teclado:

```
String frase = teclado.nextLine();
```

Foram usados também dois métodos da classe `String`. O `length` que permite obter o tamanho da **String** ou o seu número de carateres, e `charAt` que retorna o caratere da posição especificada. O ciclo percorre todos os carateres da frase, de 0 até ao seu tamanho (`length`). Lembre-se que uma **String** é um array de carateres como está representado na seguinte figura:




```
for (int i = 0; i < frase.length();++i){
    char corrente = frase.charAt(i);
```

Caso o caratere da **String** seja `r`, é adicionado à variável `transformada` um `l`, caso contrário é adicionado o caratere original.

```
transformada += 'l';
```

Ou:

```
transformada += corrente;
```

Uma **String**, face aos tipos de dados que já foram abordados, tem várias particularidades. Uma delas é a sua comparação, que difere da forma como comparamos variáveis do tipo inteiro, caratere e outros tipos básicos.

- **Crie** uma nova **classe** de nome `ComparacoesStrings.java`

```
import java.util.Scanner;
public class ComparacoesStrings {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);

        System.out.println("Insira o nome do seu pais em minusculas");
```

```
        String pais = teclado.nextLine();

        if(pais == "portugal"){
            System.out.println("O seu pais tem 89015Km2");
        }
        else if(pais == "espanha"){
            System.out.println("O seu pais tem 505954Km2");
        }
        else if(pais == "inglaterra"){
            System.out.println("O seu pais tem 244820Km2");
        }
        else {
            System.out.println("Pais desconhecido...");
        }
    }
}
```

- **Compile** e **teste** a classe com vários valores

Repare como, independentemente do valor que insere, o resultado é sempre o mesmo. Isto deve-se ao facto da comparação em **Strings** funcionar de forma diferente. Para que o programa funcione da forma que estamos à espera é necessário fazer a comparação de outra maneira:

```
if(string1.equals(string2)){
```

- **Modifique** as linhas que se encontram a realçadas:

```
import java.util.Scanner;

public class ComparacoesStrings {
```

```
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);

        System.out.println("Insira o nome do seu pais em minusculas");
        String pais = teclado.nextLine();

      if(pais.equals("portugal")){
         System.out.println("O seu pais tem 89015Km2");
        }
      else if(pais.equals("espanha")){
         System.out.println("O seu pais tem 505954Km2");
        }
      else if(pais.equals("inglaterra")){
         System.out.println("O seu pais tem 244820Km2");
        }
        else {
            System.out.println("Pais desconhecido...");
        }
    }
}
```

- **Teste** e **confirme** que o programa já funciona como esperado

5. Garbage Collection

Na declaração de uma classe, é necessário invocar o método `new`, para que seja criado um novo objeto, ou instância, dessa mesma classe. E quando esse objeto deixar de ser necessário? Será útil limpar da memória os objetos que deixaram de ser utilizados, uma vez que a memória do computador não é ilimitada. Em linguagens mais antigas como é o caso do C/C++, era essa a função do método `delete`, função esta que tinha de ser chamada pelo programador.

No Java foi implementado um conceito de ***Garbage Collection*** , existente há muito noutras linguagens, que se encarrega periodicamente de "percorrer" a memória e apagar os objetos que deixaram de ser utilizados.

Daí que no Java não existe a preocupação de controlar a utilização de memória pelos objetos não utilizados, ou seja, não é necessário invocar qualquer método para remover um objeto da memória.