

Prática 4 Classes

Objetivos

- Introdução às classes
- Decomposição de complexidade através de métodos/funções

Tópicos

- Classes e objetos
- Métodos e atributos

Exercício 4.1

Implemente classes que permitam modelar as seguintes formas geométricas:

- Círculo, caracterizado pelo seu *raio*;
- Triângulo, caracterizado pela dimensão dos seus lados (*lado1*, *lado2*, *lado3*);
- Retângulo, caracterizado por *comprimento* e *altura*.

Garanta ainda as seguintes especificações:

- crie classes que representem cada uma das figuras geométricas, implementando construtores e métodos adequados para cada classe.
- adicione todos os métodos especiais importantes (`toString()`, `equals()`, `get...()`, `set...()`, ...);
- nos construtores e métodos modificadores (`set...`), verifique pré-condições adequadas: raio e lados têm de ser valores positivos e lados do triângulo têm de satisfazer a desigualdade triangular;
- implemente um método para calcular a área de cada tipo de figura (para triângulo, ver fórmula de Heron);
- implemente um método para calcular o perímetro de cada tipo de figura;
- implemente um programa que teste todas as classes e métodos criados; o programa deve criar um conjunto de figuras, especificadas pelo utilizador através de um menu, listá-las (o método `toString()` deve mostrar o tipo e características da figura), e comparar os pares de figuras do mesmo tipo (método `equals()`).

Exercício 4.2

Analise o código fornecido no ficheiro `CashRegisterDemo.java`. Complete a classe `CashRegister` de forma a guardar um vetor de produtos e apresentar no ecrã o conteúdo no seguinte formato:

Product	Price	Quantity	Total
Book	9.99	3	29.97
Pen	1.99	10	19.90
Headphones	29.99	2	59.98
Notebook	19.99	5	99.95
Phone case	5.99	1	5.99

Total value: 215.79

Exercício 4.3

Analise o código no ficheiro `SimpleCarDemo.java`. Este programa regista informação sobre carros (classe `Car`) e regista as viagens efetuadas por cada carro, acumulando à quilometragem total do carro. Cada carro é representado pelos seguintes atributos: marca, modelo, ano de matrícula, quilometragem atual.

Complete a classe `Car` e o método `listCars`, de modo a obter o seguinte resultado no ecrã:

Carros registados:

Renault Megane Sport Tourer, 2015, kms: 35356

Toyota Camry, 2010, kms: 32456

Mercedes Vito, 2008, kms: 273891

Carro 1 viajou 773 quilómetros.

Carro 1 viajou 374 quilómetros.

Carro 1 viajou 585 quilómetros.

Carro 0 viajou 875 quilómetros.

Carro 1 viajou 692 quilómetros.

Carro 2 viajou 681 quilómetros.

Carro 2 viajou 224 quilómetros.

Carro 0 viajou 100 quilómetros.

Carro 2 viajou 95 quilómetros.

Carro 2 viajou 516 quilómetros.

Carros registados:

Renault Megane Sport Tourer, 2015, kms: 36331

Toyota Camry, 2010, kms: 34880

Mercedes Vito, 2008, kms: 275407

Exercício 4.4

Estenda o programa anterior, usando agora como base o código no ficheiro `CarDemo.java`.

O programa começa por criar um vetor para registar até 10 carros, sendo os carros registados sequencialmente desde a posição 0 do vetor. Os dados de cada carro devem ser introduzidos pelo utilizador, numa única linha e de acordo com o formato indicado abaixo, até ser introduzida uma linha em branco:

Toyota Camry 2010 74500

Os dados de entrada (marca, modelo, ano, quilometragem atual) devem ser verificados de acordo com as seguintes regras: marca é composta por uma única palavra, modelo é composto por uma ou mais palavras, ano é um número inteiro positivo composto por 4 algarismos, quilometragem é um número inteiro positivo. *Sugestão: crie uma função para esta verificação; pode, por exemplo, usar expressões regulares.*

Depois de registados os carros, o programa apresenta uma listagem dos carros e depois pede ao utilizador para registar viagens associadas a um dos carros existentes usando como formato de entrada “carro:distância”. No final, deve ser apresentada novamente a listagem dos carros.

Implemente os métodos no código fornecido, tentando reproduzir o seguinte resultado:

Insira dados do carro (marca modelo ano quilómetros): Renault Megane Sport Tourer 2015 35356

Insira dados do carro (marca modelo ano quilómetros): Toyota 2010 32456

Dados mal formatados. Tente novamente.

Insira dados do carro (marca modelo ano quilómetros): Toyota Camry 2010 32456

Insira dados do carro (marca modelo ano quilómetros):

Carros registados:

Renault Megane Sport Tourer, 2015, kms: 35356

Toyota Camry, 2010, kms: 32456

Registe uma viagem no formato "carro:distância": -1:4sdf

Dados mal formatados. Tente novamente.

Registe uma viagem no formato "carro:distância": asdc

Dados mal formatados. Tente novamente.

Registe uma viagem no formato "carro:distância": 1:673

Registe uma viagem no formato "carro:distância": 1:234

Registe uma viagem no formato "carro:distância": 0:96

Registe uma viagem no formato "carro:distância": 0:478

Registe uma viagem no formato "carro:distância": 1:123

Registe uma viagem no formato "carro:distância":

Carros registados:

Renault Megane Sport Tourer, 2015, kms: 35930

Toyota Camry, 2010, kms: 33486

Prática 5 Classes

Objetivos

- Compreender os conceitos de classe e instâncias
- Aplicar metodologias orientadas a objetos

Tópicos

- Construtores, atributos e métodos
- Métodos especiais (`toString()`, `equals()`, `get...()`, `set...()`, ...)

Exercício 5.1

Crie uma classe que permita modelar uma data (class *DateYMD*).

Esta classe deve conter os seguintes métodos **estáticos**:

- um método **booleano** que indique se o **valor inteiro que represente um mês ([1;12])** é válido: `validMonth(int month)`
- um método **inteiro** que **devolva o número de dias de um determinado mês**, num determinado ano: `monthDays(int month, int year)`
- um método **booleano** que indique se um ano é bissexto: `leapYear(int year)`
- um método **booleano** que indique se uma data **composta por dia, mês e ano, é válida**: `valid(int day, int month, int year)`

(Nota: Ao desenvolver estes métodos aproveite métodos desenvolvidos em aulas anteriores.)

A **classe deve** também **permitir instanciar objetos** que representem **uma data específica (válida)**. Nesse sentido, considere que a representação interna do objeto é composta por três atributos inteiros (`day`, `month`, `year`).

Deve ser possível aplicar externamente as seguintes operações sobre objetos deste tipo:

- definir uma data: `set(int day, int month, int year)`;
- consultar os valores do dia, mês e ano (`day`, `month`, `year`);
- incrementar a data (`increment`);
- decrementar a data (`decrement`);
- método `toString` que devolva a data no formato AAAA-MM-DD.

A classe deve ter um construtor que defina uma data (válida) indicando um dia, mês e ano.

(Nota: desenvolva a classe garantindo que não é possível que nenhum dos seus objetos represente uma data inválida [por exemplo, 31 de fevereiro de 2022].)

Para testar esta classe, crie um programa de teste, com o seguinte menu:

Date operations:

- 1 - create new date
- 2 - show current date
- 3 - increment date
- 4 - decrement date
- 0 - exit

Exercício 5.2

Construa uma classe que represente um calendário. Esta classe deve permitir registar o número de eventos agendados para cada dia do ano, sugerindo-se para isso o uso de um vetor bidimensional de inteiros.

A classe deve fazer uso da classe *DateYMD* desenvolvida no exercício anterior, e deve incluir:

- um construtor que recebe o ano e o dia da semana (entre 1-domingo e 7-sábado) em que começa o ano;
- métodos que devolvem esses dados (consultas/getters): `year()` e `firstWeekdayOfYear()`;
- um método que devolva o dia da semana em que começa um dado mês (no ano do calendário): `firstWeekdayOfMonth(month)`;
- métodos que permitam adicionar/remover um evento numa data: `addEvent(DateYMD)`; `removeEvent(DateYMD)`;
- método `toString` que devolva a representação de um mês de calendário: `printMonth(month)`; nesta representação, cada dia deve ser precedido de * caso exista pelo menos um evento agendado nessa data;
- método `toString` que devolva o calendário para todo o ano:

```
      January 2023
Su  Mo  Tu  We  Th  Fr  Sa
  1   2   3   4   5   6   7
  8   9  10  11  12  *13  14
 15  16  17  18  19  20  21
 22  23  24  25  26  27  28
 29  30  31
```

```
      February 2023
Su  Mo  Tu  We  Th  Fr  Sa
      1   2   3   4
*5   6   7   8   9  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28
```

...

Para testar esta classe, crie um programa de teste, com o seguinte menu:

```
Calendar operations:
1 - create new calendar
2 - print calendar month
3 - print calendar
0 - exit
```

Exercício 5.3

Defina uma classe para representação de imóveis numa agência de leilões imobiliários. Cada imóvel pode ser colocado em leilão numa determinada data e por um período variável.

Os imóveis têm um identificador numérico sequencial, começando em 1000, e são caracterizados pelo número de quartos, localidade, preço, disponibilidade (booleano), e pelas datas de início e final do leilão, quer poderão ser ambas nulas.

Defina também uma classe para representar a agência de leilões, que terá métodos para gerir os imóveis e os leilões.

Usando o ficheiro Ex3.java como base, implemente as classes descritas acima de forma a obter o seguinte resultado no ecrã:

```
Imóvel 1001 vendido.  
Imóvel 1001 não está disponível.  
Imóvel 1010 não existe.  
Propriedades:  
Imóvel: 1000; quartos: 2; localidade: Glória; preço: 30000; disponível: sim  
Imóvel: 1001; quartos: 1; localidade: Vera Cruz; preço: 25000; disponível: não  
Imóvel: 1002; quartos: 3; localidade: Santa Joana; preço: 32000; disponível: sim; leilão  
2023-03-24 : 2023-03-24  
Imóvel: 1003; quartos: 2; localidade: Aradas; preço: 24000; disponível: sim; leilão 2023-  
04-03 : 2023-04-03  
Imóvel: 1004; quartos: 2; localidade: São Bernardo; preço: 20000; disponível: sim
```