Exercício 1

Num projeto: `2016.pp.fp08` com um *package* `ObjectManagement`, implemente uma classe: `ContainerOfObjects` que permita realizar a gestão de objetos (`Object`) num vetor. Esta classe deverá permitir a sua instanciação de três formas diferentes:

- A primeira definindo um valor máximo de elementos que a mesma poderá possuir;
- A segunda com um valor por defeito à sua escolha (tamanho omitido);
- A terceira instanciando com base num vetor de elementos recebido.

Deverá ainda permitir as operações básicas de adicionar, remover, substituir e procurar objetos. Toda a gestão de como os elementos são guardados e trabalhados dentro desta classe deve estar encapsulada e apenas visível para as suas subclasses.

(Exemplo) Resolução parcial:

```
public class ContainerOfObjects {

    private final int DEFAULT_SIZE = 100;
    private Object objects[];

    /**
     * Construtor que permite a instanciação da classe tendo por
     * base um vetor de elementos recebido
     *
     * @param objects Lista de objetos sem tamanho fixo
     */
    public ContainerOfObjects(Object[] objects) {...3 lines }

     /**
     * Construtor que permite a instanciação da classe tendo por base
     * um valor por defeito (100)
     *
     */
    public ContainerOfObjects() {...3 lines }

      /**
     * Construtor que permite a instanciação da classe definindo
     * um valor máximo de elementos
     *
     * @param maxSize número máximo de elementos permitidos no vetor
     */
    public ContainerOfObjects(int maxSize) {...3 lines }
```

Figura 1. Excerto dos métodos construtores da classe: `ContainerOfObjects`

```java
/**
 * Método responsável por inserir um {@link Object objeto} na coleção de
 * {@link ContainerOfObjects#objects objetos}
 *
 * @param newObject {@link Object objeto} a inserir no vetor
 * @return valor booleano que sinaliza o sucesso/insucesso da operação
 */
protected boolean addObject(Object newObject) {...9 lines }

/**
 * Método responsável por remover um {@link Object objeto} do vetor de
 * {@link ContainerOfObjects#objects objetos}
 *
 * @param position indice correspondente ao elemento a eliminar
 *
 * @return o {@link Object objeto} eliminado
 */
protected Object removeObject(int position) {...7 lines }

/**
 * Método responsável por substituir um {@link Object objeto} no vetor de
 * {@link ContainerOfObjects#objects objetos}
 *
 * @param position indice correspondente ao elemento a modificar
 * @param newObject novo objeto a colocar no vetor
 *
 * @return valor booleano que sinaliza o sucesso/insucesso da operação
 */
protected boolean setObject(int position, Object newObject) {...9 lines }

/** Método responsável por encontrar um {@link Object objeto} no vetor de ...9 lines */
protected int findObject(Object obj) {...8 lines }
```

Figura 2. Excerto de alguns métodos existentes na classe: `ContainerOfObjects`

Exercício 2

Tendo por base o exercício 1 da ficha prática 7, crie uma classe `BicycleSalesManagement` que suporte a gestão de vendas de uma loja. Esta classe deverá possuir:

- Identificação da venda (`saleID`);
- Dia, mês e ano da venda;
- Capacidade para gerir (adicionar, remover, substituir e procurar) uma coleção de bicicletas associadas à venda;
- Total da venda (o preço total depende do preço de cada bicicleta contida no vetor de bicicletas).

2.1. Tendo por base a classe `ContainerOfObjects` implementada no exercício anterior, implemente os métodos necessários para que possa responder aos requisitos pedidos.

Resolução parcial:

```
public class BicycleSalesManagement extends ContainerOfObjects {

    private int SaleID;
    private SaleDate data;
    private double total;

    /** Método construtor que permite inicializar a venda com um conjunto de ...8 lines */
    public BicycleSalesManagement(Bicycle[] objects, int SaleID, SaleDate data) {...5 lines }

    /** Método construtor que permite a instanciação do vetor de bicicletas com o ...8 lines */
    public BicycleSalesManagement(int maxSize, int SaleID, SaleDate data) {...5 lines }

    /**
     * Método responsável por inserir uma {@link Bicycle bicicleta} na coleção
     * de {@link ContainerOfObjects#objects objetos}
     *
     * @param bike {@link Bicycle bicicleta} a inserir no vetor
     *
     * @return valor booleano que sinaliza o sucesso/insucesso da operação
     */
    public boolean addBicycle(Bicycle bike) {...3 lines }

    /**
     * Método responsável por remover uma {@link Bicycle bicicleta} do vetor de
     * {@link ContainerOfObjects#objects objetos}
     *
     * @param bike {@link Bicycle bicicleta} a eliminar
     *
     * @return a {@link Bicycle bicicleta} eliminada
     */
    public Bicycle removeBicycle(Bicycle bike) {...8 lines }
```

Figura 3. Excerto de alguns métodos existentes na classe: `BicycleSalesManagement`

2.2. Crie a classe `BikeStoreDemo` de forma a testar e a imprimir a informação que pode extrair de cada venda realizada: data da venda, lista bicicletas compradas e o preço final a pagar.
Tenha em atenção as seguintes questões sobre o teste que realizou nas classes implementadas:

- O método `removeBicycle` funcionou corretamente?
- Como é que o método `findObject` da classe `ContainerOfObjects` utilizado no método `removeBicycle` da classe `BicycleSalesManagement` realiza a comparação dos objetos?
- Implemente as alterações necessárias para corrigir o problema (DICA: método `equals`[1] da classe `Object`);
- Implemente o método `hasObject` na classe `ContainerOfObjects` que seja capaz de determinar se o objeto recebido por argumento está no seu vetor. Reescreva o método `equals` na classe `Bicycle`.

2.3. Implemente um método: `printAll` na classe `containerOfObjects` que imprima todos os dados dos objetos armazenados no vetor. De seguida, implemente um método: `printAllBicycles` na classe `BicycleSalesManagement` utilizando o método `printAll`. Para isso, deverá reescrever o método `toString()`[1] herdado da classe `object` na classe `Bicycle`. Este método deverá ser responsável devolver uma *String* com todos os dados genéricos de uma bicicleta. Note que poderá ter de implementar o método `toString()` nas restantes subclasses de `Bicycle`.

2.4. No contexto do exercício de gestão de uma loja de bicicletas, são utilizadas várias coleções para gerir os diversos componentes de negócio. Para além das vendas, a classe `DeliveryBike` possui um vetor para suportar a gestão de patrocínios (`Sponsors`) e a classe `MountainBike` possui um vetor para suportar a gestão de ferramentas/utensílios (`BikeTools`). Implemente as classes: `BikeToolsManagement` e `SponsorsManagement` conforme apresentado nas Figuras 3 e 4.

[1] https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

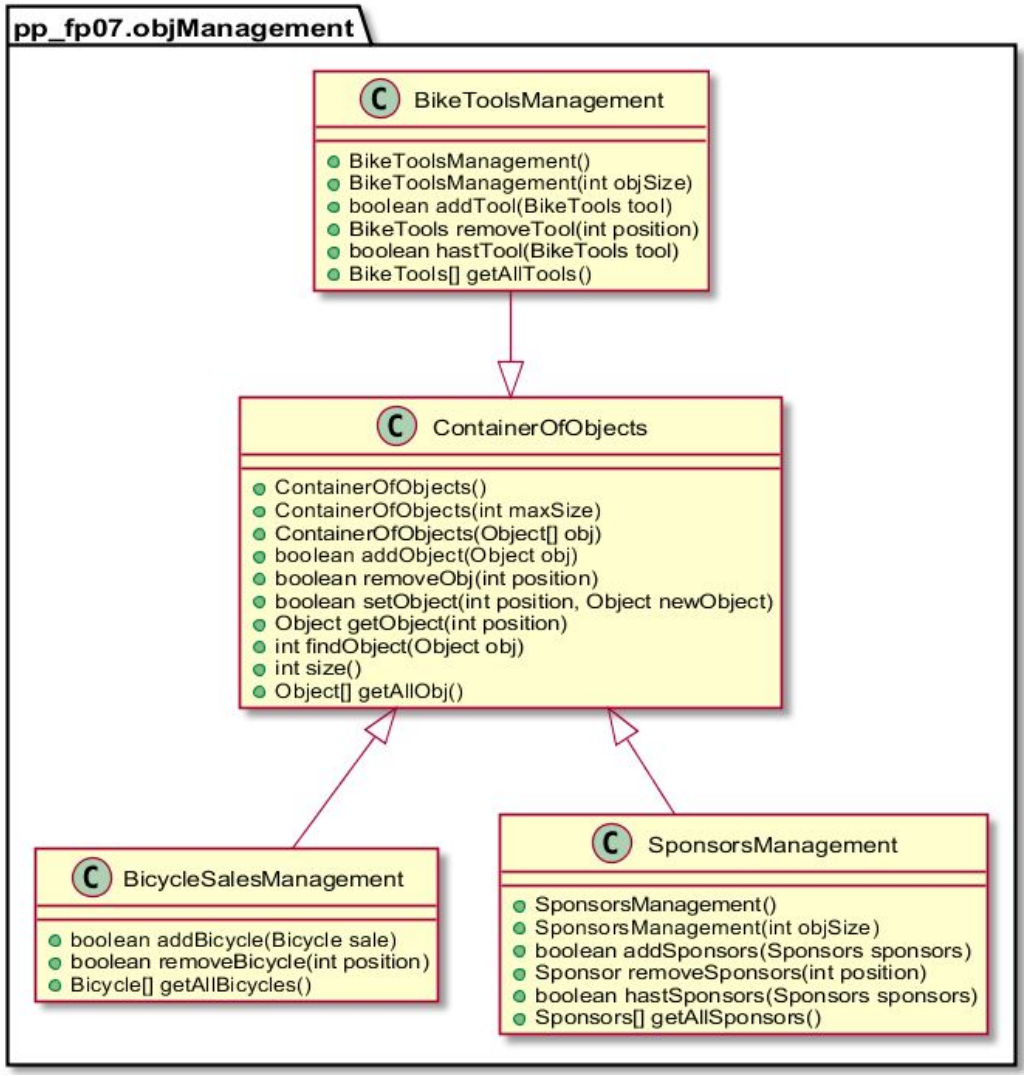

Figura 3. Excerto das classes `ContainerOfObjects`, `BicycleSalesManagement`, `SponsorManagement` e `BikeToolsManagement`.