Sumário

- Coleções de Objetos

Documentação complementar Java

Nota: A resolução desta ficha prática pressupõe a utilização da ferramenta git para ajudar na gestão de versões do software desenvolvido.

Documentação complementar Git:

- learngitbranching
- gitexplorer

Exercício 1

Num projeto: 2019.pp.fp08 vamos criar um projeto com um package ObjectManagement, que implemente uma classe, ContainerOfObjects, baseada na resolução parcial disponível nas figuras 1 e 2, que permita realizar a gestão de objetos do tipo 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.

Ao iniciar este projeto deverá também iniciar também um repositório git na pasta do projeto correndo o comando:

- `git init`

Verifique que na raiz do projeto é criada a pasta .git, que sinaliza o uso de um repositório git. Durante o desenvolvimento de um projeto em Netbeans existem pastas/ficheirios que não devem ser submetidos para o repositório git. Podemos excluir esses elementos usando um ficheiro com o nome .gitignore armazenado na pasta do projeto. Antes do primeiro commit para o repositório, crie este ficheiro com o seguinte conteúdo.

```
# NetBeans specific #
nbproject/private/
build/
dist/
# Class Files #
*.class
# Package Files #
*.jar
*.war
*.ear
```

Ficheiro 1 - Exemplo ficheiro ".gitignore"

Neste ficheiro estamos a excluir todos os ficheiros do repositório git mencionados no seu conteúdo. Estes são um conjunto de ficheiros que por boa prática não devem ser associados ao repositório git. Podemos agora fazer o primeiro commit ao projeto e todos commit subsequentes irão excluir sempre os ficheiros e

pastas declarados no ficheiro “.gitignore”

```
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 - Resolução parcial da classe ContainerOfObjects

Após a resolução do exercício deve gerar a o javadoc a partir dos comentários e responda às seguintes questões:

- A classe implementada pode funcionar/armazenar objetos de qualquer tipo?
- Porque foi inicializado um array de objetos do tipo Object para guardar os objetos?

```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 - Resolução parcial da classe ContainerOfObjects (continuação)

Exercício 2

Tendo por base o exercício 1 da ficha prática 7, e usando o repositório git criado na última aula prática, realize um clone do repositório criado no github. Na pasta do projeto crie o ficheiro .gitignore e use o exercício da última aula na resolução deste exercício. A cada alínea deste exercício faça o respectivo commit para o repositório git.

Crie uma classe BicycleSalesManagement que suporte a gestão de vendas de uma loja. Verifique a resolução parcial disponível na figura 3. 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.

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 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 .

```
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
     * do {@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 - Resolução parcial da classe: BicycleSalesManagement

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 “RoadBike” possui um vetor para suportar a de observações e a classe MountainBike possui um vetor para suportar a gestão de ferramentas/utensílios (BikeTools). Implemente as classes: BikeToolsManagement e CommentsManagement (Figura 4). Considere que:

- Não podem existir BikeTools iguais para uma MountainBike;
- Cada observação de uma RoadBike deverá ter no máximo 60 caracteres.

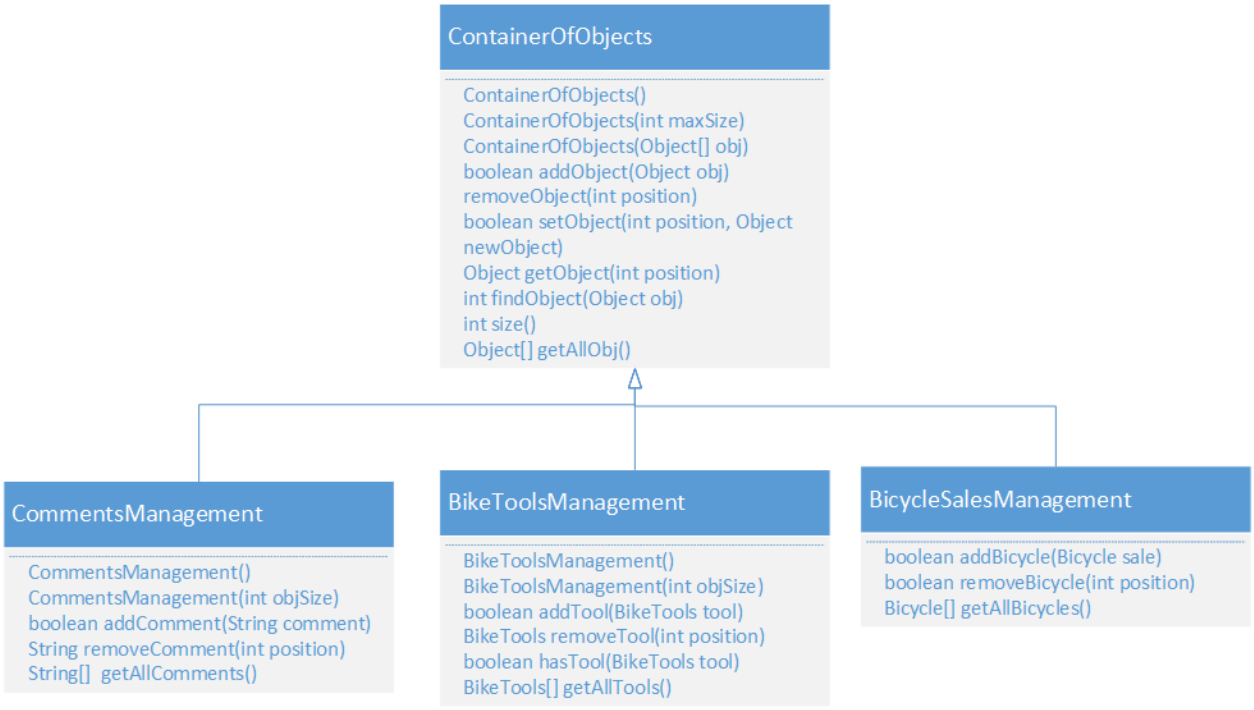

Figura 4 - Excerto das classes ContainerOfObjects, BicycleSalesManagement, SponsorManagement e BikeToolsManagement.