PRÁTICA LABORATORIAL 11

Objetivos:

- Classes Abstratas

EXERCÍCIOS

Parte 1

1. Desenvolva uma pequena aplicação que permita armazenar informação relativa a um conjunto de bicicletas disponíveis para venda numa loja especializada. A loja comercializa essencialmente bicicletas de dois tipos: bicicletas de montanha e de carga/distribuição. As bicicletas possuem: identificador (id), número de velocidades (numberOfGears), cor principal (mainColor), diâmetro das rodas (wellSize) e comprimento (bikeLenght). Adicionalmente, podem possuir um assento regulável (adjustableSeatpost) e o preço (price). Para além dos dados já mencionados, cada tipo bicicleta possui características próprias:
    a. A Bicicleta de montanha possui um número de luzes (numberofLights), tipo de suspensão (suspension) (p.ex. suspensão simples, dupla, ou sem suspensão) e um conjunto de utensílios/acessórios (bikeTools) aplicável apenas a este tipo de bicicleta (p.ex. garrafa de água, kit para substituição do pneu, etc). A bicicleta de montanha não possui um limite de utensílios.
    b. A Bicicleta de carga/distribuição possui a localização (isFrontBasket), a capacidade do cesto (basketDelivery) e um conjunto de máximo de 10 patrocínios (Sponsors).

Resolução parcial:

```java
/**
 * Método construtor para a criação de uma instância de
 * {@link Biclycle bicicleta}.
 *
 * @param id Identificador da bicicleta
 * @param numberOfGears Número de velocidades
 * @param mainColor Cor da bicicleta
 * @param weelSize Diâmetro das rodas
 * @param bikeLenght Comprimento da bicicleta
 * @param adjustableSeatpost Saber se assento é ajustável
 * @param price Preço da bicicleta
 */
public Bicycle(int id, int numberOfGears, String mainColor, float weelSize,
        float bikeLenght, boolean adjustableSeatpost, float price) {
    this.id = id;
    this.numberOfGears = numberOfGears;
    this.mainColor = mainColor;
    this.weelSize = weelSize;
    this.bikeLenght = bikeLenght;
    this.adjustableSeatpost = adjustableSeatpost;
    this.price = price;
}

/**
 * Retorna o {@link Bicycle#id id} de uma bicicleta
 * @return
 */
public int getId() {
    return id;
}
```

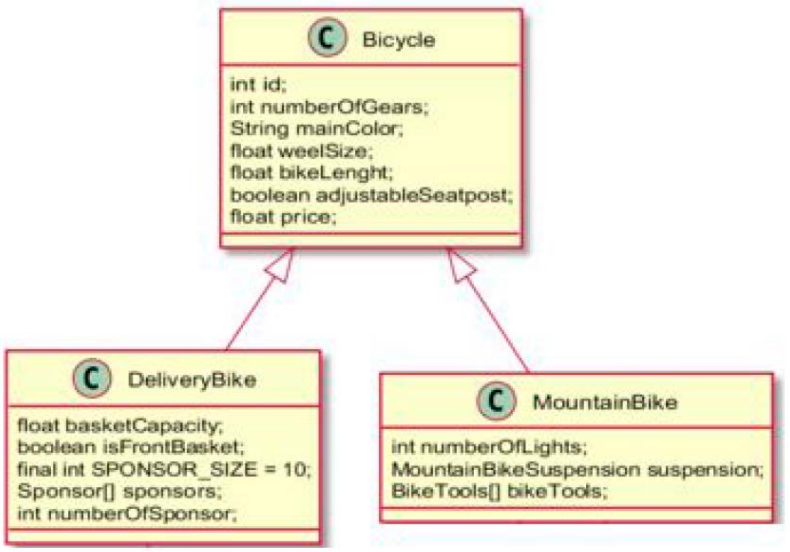
Método construtor e excerto dos métodos de acesso da classe `Bicycle`

```java
public class MountainBike extends Bicycle{
    /**
     * Número de luzes
     */
    private int numberOfLights;
    /**
     * Tipo de suspensão
     */
    private MountainBikeSuspension suspension;
    /**
     * Utensílios existentes neste tipo de bicicleta
     */
    private BikeTools[] bikeTools;

    /**
     * Método construtor para a criação de uma instância de
     * {@link MountainBike bicicleta de montanha} sem utensílios.
     *
     * @param numberOfLights Número de luzes
     * @param suspension Tipo de suspensão
     * @param id Identificador da bicicleta
     * @param numberOfGears Número de velocidades
     * @param mainColor Cor da bicicleta
     * @param weelSize Diâmetro das rodas
     * @param bikeLenght Comprimento da bicicleta
     * @param adjustableSeatpost Saber se assento é ajustável
     * @param price Preço da bicicleta
     */
    public MountainBike(int numberOfLights, MountainBikeSuspension suspension,
            int id, int numberOfGears, String mainColor, float weelSize,
            float bikeLenght, boolean adjustableSeatpost, float price) {
        super(id, numberOfGears, mainColor, weelSize, bikeLenght, adjustableSeatpost,
                price);
        this.numberOfLights = numberOfLights;
        this.suspension = suspension;
    }
```

Excerto da classe `MountainBike`

Num projeto poo_fp11, com um package poo_fp11.bikeStore, implemente o código Java necessário para representar a estrutura descrita. Tenha por referência o seguinte excerto da estrutura de classes:



c. Deve garantir o encapsulamento de todas as classes criadas;
d. Deve criar métodos de acesso necessários para todas as classes criadas;
e. Para as coleções de ferramentas (BikeTool) e patrocinadores (Sponsors), deverá criar métodos que permitam o acesso controlado a cada uma das coleções. Por exemplo, deverá criar métodos de inserção, edição, remoção e listagem de elementos.

f. Crie a classe BikeDemo e teste as classes implementadas, inicializando alguns elementos relativo a bicicletas de montanha e bicicletas de carga/distribuição.

g. Crie uma classe BicycleManagment que armazene um vetor de bicicletas (Bicycle). Crie um método para adicionar bicicletas ao ArrayList. De seguida e utilizando a classe BikeDemo ,teste a classe criada, adicionando as bicicletas que criou na alínea a. Crie ainda um método na classe BicycleManagment que imprima todos os dados das bicicletas contidas no vetor que criou.

2. Crie uma classe abstrata chamada "Animal" que tenha um método abstrato "fazerBarulho()". Em seguida, crie cinco subclasses de "Animal" chamadas "Cão", "Gato", “Vaca”, “Porco” e “Galinha”, cada uma implementando o método "fazerBarulho()" para que imprima "Au au au", "Miau", “Muuuuu”, “Oinc oinc” e “Pi pi pi”, respetivamente. Crie também uma classe Farm que tenha uma lista de animais e um método makeNoise que faz com que todos os animais da lista emitam o seu som. Teste a classe, instanciando uma nova quinta com, pelo menos, 5 animais. De seguida, invoque o método makeNoise.

3. Transforme a Classe Carro numa Classe Abstrata denominada Veículo. De seguida, crie a classe Carro, a classe Mota e a classe Camião devendo herdar os atributos e comportamentos de veículo.

   a. Classe Veículo: Deve ter atributos de Marca, Modelo, Ano de Fabrico, Potência, Cilindrada e litros100Km. Os métodos deverão ser os previamente implementados, ou seja, ligar, corrida e calcularConsumo.

   b. Classe Carro: Para além de se assumir um veículo, deve também ter o atributo tipoCombustivel (Enumeração). Elabore o método calcularCusto que com base no tipoCombustivel deve apresentar em € o custo da viagem (pode fazer uso de outros métodos). (GASOLINA = 1.95€/L, DIESEL = 1.75€/L, GPL = 0.95€/L).

   c. Classe Mota: Para além de se assumir um veículo, deve também ter o método imagem que imprime o conteúdo do ficheiro mota.txt na consola.

   d. Classe Camião: Para além de se assumir um veículo, deve ter o atributo capacidadeCarga (em Kg). Elabore também o método viagem que recebe como parâmetro a distância e a carga (em Kg), avalie se o camião tem capacidade para essa carga, caso ultrapasse a capacidade deve recursar a viagem. Caso a carga esteja dentro da capacidade, deve calcular o consumo e custo da viagem, sabendo que cada 100Kg de carga aumentam 0,2L/100Km ao consumo.

   e. Teste as classes instanciando um veículo de cada tipo. Faça uma corrida entre dois carros e imprima os dados do vencedor. Calcule o custo de uma viagem do carro. E faça uma viagem válida e outra inválida (excesso peso) para o camião.

4. Desenvolva uma aplicação que permita gerir pizzas de uma pequena pizzaria. Cada pizza disponibilizada pelo restaurante é composta por um conjunto de ingredientes associados (no máximo cada pizza terá 5 ingredientes) e respetivas quantidades. Cada ingrediente é identificado pelo seu código, nome, unidade de medida (por exemplo: Gramas, Litros ou Unidades) e o número de calorias associadas. Cada pizza é caracterizada pelo seu código, nome, descrição, preço, tamanho (Pequena, Média ou Grande), número de ingredientes e uma coleção de ingredientes que representam a composição da pizza.
    a. Num package poo_fp11.PizzaRestaurant , crie as classes necessárias para responder aos requisitos do problema, considerando que:
    b. Deve garantir o encapsulamento de todas as classes criadas;
    c. Deve criar métodos de acesso necessários para todas as classes criadas;
    d. Num package poo_fp11.PizzaRestaurant.Enums , deve criar as enumerações para representar a unidade do ingrediente (Gramas, Litros ou unidades) e para o tamanho da pizza (Pequena, Média ou Grande). A impressão da unidade dos ingredientes e do tamanho da pizza deve ser apresentada com uma mensagem descritiva (exemplo na Figura 1);
    e. Utilize a palavra reservada this para se referir a cada variável de instância em todas as classes criadas.

```
public enum PizzaSize {
    SMALL, MEDIUM, BIG, KING;

    public static String PizzaSizeToString(PizzaSize size) {
        switch (size) {
            case SMALL:
                return "The pizza is small.";
            case MEDIUM:
                return "The pizza is medium.";
            case BIG:
                return "The pizza is big.";
            case KING:
                return "The pizza is king size.";
            default:
                return "The pizza is medium.";
        }
    }
}
```

Figura 1: Exemplo de enumeração

Resolução parcial:

```
public class Pizza {
    private final int MAX_INGREDIENTS = 5;
    private int id;
    private String name;
    private String description;
    private float price;
    private PizzaSize size;
    private PizzaIngredients[] ingredients;
    private int numberOfIngredients;

    /**
     * Método construtor para a criação de uma instância de
     * {@link Pizza pizza}.
     *
     * @param id Código que identifica uma <b>Pizza</b>
     * @param name Nome da <b>Pizza</b>
     * @param description Descrição resumida da <b>Pizza</b>
     * @param price Preço da <b>Pizza</b>
     * @param size Tamanho associado à <b>Pizza</b>
     */
    public Pizza(int id, String name, String description, float price, PizzaSize size) {
        ingredients = new PizzaIngredients[MAX_INGREDIENTS];
        numberOfIngredients = 0;
        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
        this.size = size;
    }
```

Figura 2: Excerto da classe Pizza

f. Teste as classes criadas através de uma classe PizzaDemo. Crie no mínimo, duas pizzas (deverá utilizar uma coleção) com pelo menos três ingredientes cada.

g. Na classe Pizza, adicione métodos que permitam:

   i. Associar novos ingredientes a uma Pizza até a um máximo de 5. Não deve ser permitido aceder diretamente à variável que representa a coleção dos ingredientes da pizza;

   ii. Editar a quantidade de um ingrediente que pertença à coleção de ingredientes de uma pizza;

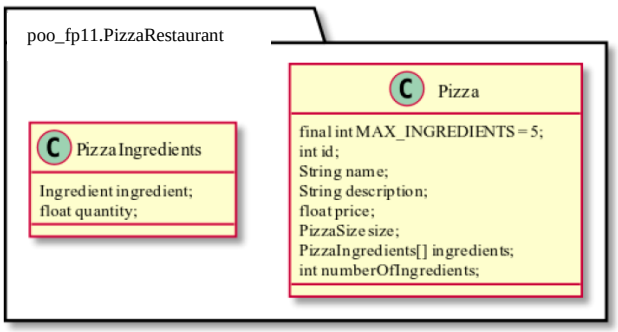
   iii. Remover um ingrediente (identificando o ingrediente pelo seu id) que pertença à coleção de ingredientes;

   iv. Determinar o número calorias da Pizza;

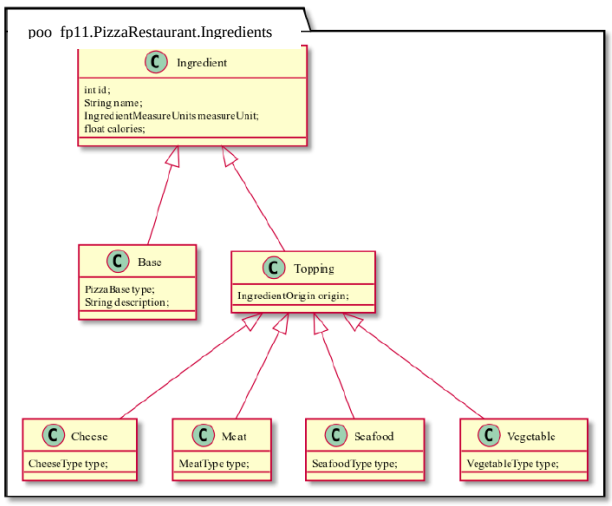
   v. Apresentar uma descrição detalhada da pizza, assim como dos seus ingredientes.

h. Teste as alterações na classe PizzaDemo .

5. (Parte 2) Considere que cada ingrediente de uma pizza é identificado pelo seu código, nome, unidade de medida (por exemplo: Gramas, Litros ou Unidades) e o número de calorias associadas. No entanto, é necessário que uma pizza possua (pelo menos) dois tipos de ingredientes:
    a. Base, que descreve o tipo de massa utilizada (Massa alta ou massa fina);
    b. Cobertura da pizza que engloba alguns tipos de ingredientes e os seus tipos:
        i. Queijo (Mozzarela, Serra, etc);
        ii. Carne (Porco, Vaca, Salame, etc);
        iii. Vegetal (Tomate, Cebola, Cogumelos, etc);
        iv. Frutos do mar(Camarão, Lagosta, etc).
    c. Como os clientes da pizzaria são muito exigentes em relações aos produtos que são utilizados para confecionar a pizza, qualquer tipo de ingrediente que faça parte da cobertura da Pizza possui um atributo que permite identificar a sua origem (nacional ou importado).
    d. Implemente as alterações necessárias de modo que possa refletir no exercício anterior todos os requisitos apresentados. Tenha por base o seguinte diagrama para o auxiliar na resolução do exercício:



Excerto da estrutura para representação das classes `Pizza` e `PizzaIngredients`



Excerto da estrutura para representação das classes e respetivas subclasses

Resolução parcial:

```
public class Meat extends Topping {
    private MeatType type;

    /**
     * Método construtor para a criação de uma instância de {@link Meat meat}
     *
     * @param id Identificação de um {@link Ingredient ingrediente}
     * @param name Nome do ingrediente {@link Ingredient ingrediente}
     * @param measureUnit {@link IngredientMeasureUnits Unidade}
     * de medida do {@link Ingredient ingrediente}
     * @param calories Número de calorias associado a um
     * {@link Ingredient ingrediente}
     * @param type Tipo de {@link MeatType carne}
     * @param origin {@link IngredientOrigin Origem} da carne
     */
    public Meat(int id, String name, IngredientMeasureUnits measureUnit,
                float calories, MeatType type, IngredientOrigin origin) {
        super(origin, id, name, measureUnit, calories);
        this.type = type;
    }

    /**...3 lines */
    public MeatType getType() {...3 lines }

    /**...3 lines */
    public void setType(MeatType type) {...3 lines }

}
```

Figura 2: Excerto da classe `Meat`

e. Realize as alterações necessárias para que a unidade de medida por defeito dos ingredientes do tipo PizzaBase seja obrigatoriamente gramas.

f. Altere a classe PizzaDemo de forma a testar as alterações realizadas. Crie pelo menos um ingrediente de cada tipo.

g. Na classe Pizza, adicione/altere métodos que permitam:

   i. Que apenas seja possível adicionar ingredientes do tipo Topping quando a pizza já possuir um ingrediente do tipo Base;

   ii. Não deverá ser possível ter mais do que um ingrediente do tipo Base;

   iii. No máximo, a pizza deverá possuir 5 ingredientes do tipo Topping ;

   iv. Defina um tipo da pizza tendo por base dos ingredientes que esta possui, considerando a seguinte classificação:

      1. Pizza de carne: Contém apenas ingredientes do tipo Meat.
      2. Pizza do mar: Apenas ingredientes do tipo Seadfood;
      3. Pizza vegetariana: Apenas ingredientes do tipo Vegetable;

h. Teste as alterações na classe PizzaDemo. Crie um método para imprimir o conteúdo (todos os atributos) de todos os ingredientes de uma pizza. **Bom trabalho!** ☺