MC322 – Programação Orientada a Objetos
Laboratório 13 – 2s2021
TESTE PRÁTICO 4

Leonardo Montecchi (Professor) Thales Eduardo Nazatto Ângelo Renato Pazin Malaguti
Leonardo de Sousas Rodrigues

Para perguntas ou dúvidas usem o Discord (`https://discord.gg/Xzpz9epNTG`)

1 Submissão

Data de entrega

- **21/11/2021** até às **23h59**.

Formato de Submissão

- Ao criar o projeto Java, selecionar a versão **JavaSE-11** no JRE
- IMPORTANTE: Nomear o projeto na forma **RA_Lab13** e o pacote base na forma **com.unicamp.mc322.lab13**. Substitua RA com o seu *Registro Acadêmico* (matrícula).
- Submeta o trabalho no link de entrega na página do Classroom da disciplina, em formato de arquivo compactado (zip) com o nome **RA_Lab13**. Substitua RA com o seu *Registro Acadêmico* (matrícula). Entregas feitas de outras formas não serão consideradas.
- O arquivo compactado deve conter o **projeto inteiro** ("File / Export / Archive File/ Select Project" se usar o Eclipse, ou crie o arquivo compactado manualmente).

Critérios de avaliação

- Este laboratório **vale** nota.
- Em particular, o código será avaliado de acordo com os seguintes aspectos:
  1. Aplicação de princípios de POO (60%).
     - Ex: *encapsulamento* de atributos, *responsabilidades* de classes, *Exceções*, *Interfaces*
  2. Funcionalidades implementadas (40%).

2 Tarefa

Lembrando: O foco do laboratório não é no desempenho das operações implementadas, mas em como você aplica os conceitos de programação a objetos abordados em aula.

2.1 Contexto

Um novo empreendedor entra no mercado oferecendo sorvetes. Para isso, o empresário aluga um imóvel em uma rua principal, contrata uma empresa X1 para o serviço de entrega e contrata uma empresa X2 para divulgar os seus produtos na Internet. Após organizar as instalações do local físico e implementar o desenho de interiores, o empreendedor observou que precisa de **uma aplicação para administrar as prioridades dos pedidos presenciais ou feitos pela Internet**. Desta maneira, os funcionários dele poderão saber qual pedido atender primeiro.

2.2 Especificações

2.2.1 Sumário

- A aplicação precisará de uma estrutura de dados personalizada que possa gerenciar os pedidos. Esta estrutura de dados deverá implementar a interface **ICrazyDS** (As operações da interface são definidas em Seção 2.2.3). ***Dica:*** *Você pode implementar a estrutura de dados reusando classes das bibliotecas do Java (p. ex., LinkedList) dentro dela.*
- Há duas categorias de pedidos: Pedidos feitos na loja física e pedidos feitos pela Internet. Ambas as categorias implementam a interface **IOrder** (As operações da interface são definidas em Seção 2.2.2).
- Há estratégias diferentes para organizar os pedidos. Uma definida pelo empreendedor e outra que você deverá definir. Cada estratégia deve implementar a interface **IOrderingStrategy** (As operações da interface são definidas em Seção 2.2.4).

2.2.2 Pedidos

Todo pedido registra a informação de quem fez o pedido (p.ex., uma pessoa do Brasil). Generalmente uma pessoa do Brasil tem nome, CPF e idade. Um pedido (seja feito no local físico ou pela Internet) deve implementar a interface **IOrder**. Entre as suas informações, um pedido registra o número de turnos que o pedido ficou esperando para ser atendido.

Operações da IOrder interface

- Uma operação para incrementar o número de turnos que o pedido ficou esperando para ser atendido.
- Uma operação para obter o número de turnos que o pedido ficou esperando para ser atendido.
- Uma operação para obter o código do pedido (p. ex., AWD_2021_ABCD).
- Uma operação para fazer a impressão reduzida no console de quem fez o pedido (p. ex., nome).
- Uma operação para fazer a impressão completa no console de quem fez o pedido (p. ex., nome, CPF e idade).
- Uma operação para obter a pessoa que fez o pedido.

A interface **IOrder** pode ter outras operações que você achar necessárias.

2.2.3 Estrutura de dados

A estrutura de dados (p. ex., *MyCustomDS*) deve implementar a interface **ICrazyDS**.

Operações da ICrazyDS interface

- Adicionar um elemento IOrder. O elemento adicionado deverá ocupar a sua posição na Fila de prioridade tomando em consideração a estratégia definida na estrutura de dados.
- Remover um elemento IOrder da estrutura de dados usando o elemento IOrder como entrada.
- Obter o elemento IOrder que atualmente tem a maior prioridade na estrutura de dados sem retirá-lo da estrutura (*peek*).

- Fazer a impressão dos elementos IOrder da estrutura (pedidos) por ordem de prioridade decrescente (o primeiro é o que tem maior prioridade).

A interface **ICrazyDS** pode ter outras operações que você achar necessárias.

2.2.4 Estratégias

Você deverá implementar duas diferentes estratégias para definir a prioridade dos elementos IOrder (pedidos) da sua estrutura de dados.

- Uma estratégia baseada no PriorityScore, definido abaixo.
- Uma estratégia qualquer da sua escolha.

Cada uma das estratégias deve implementar a interface **IOrderingStrategy**.

Operações da IOrderingStrategy interface

- Calcular a pontuação de prioridade do pedido dado o elemento IOrder (pedido).

A interface **IOrderingStrategy** pode ter outras operações que você achar necessárias.

PriorityScore Uma das estratégias que o empreendedor pensa que possa funcionar bem no seu empreendimento é definir a prioridade com base na idade da pessoa que fez o pedido e o tempo de espera do pedido. Esta estratégia, chamada de *PriorityScore*, está descrita na Equação 1, onde **PersonAge(p)** representa a idade da pessoa que fez o pedido **p** e **Shifts(p)** representa a quantidade de turnos que o pedido **p** esperou na Fila. Desta maneira, a estrutura de dados pode usar a **PriorityScore** (pontuação obtida) do pedido **p** para organizar os seus elementos.

$$\textbf{PriorityScore (p)} = \underbrace{\frac{\textbf{PersonAge (p)}}{100}}_{\text{Part that considers the Age}} + \overbrace{\frac{7}{100} \times \textbf{Shifts(p)}}^{\text{Part that considers the Shifts}} \tag{1}$$

2.2.5 Exceções

Você deverá selecionar quais métodos das interfaces definidas e das suas classes lançarão uma exceção. Cada categoria de exceção que você defina deverá ser personalizada (p.ex., CrazyDSException).

2.3 Fluxo de Execução

Considere o seguinte cenário como um exemplo de fluxo de execução. Complemente e ajuste o fluxo para mostrar as características que você implementou. Os nomes das classes no fluxo são apenas referências e você poderá mudá-las caso deseje.

```
public class Main {
    public static void main(String[] args) {
        ICrazyDS crazyDS = new CrazyDS(new YourStrategy1());
        //ICrazyDS crazyDS = new CrazyDS(new YourStrategy2());
        IOrder order1 = new InternetOrder(new PersonPT(LocalDate.of( 1985 , Month.JANUARY , 1 ), "CPF1", "name1"));
        IOrder order2 = new InternetOrder(new PersonPT(LocalDate.of( 1986 , Month.JANUARY , 2 ), "CPF2", "name2"));
        IOrder order3 = new InternetOrder(new PersonPT(LocalDate.of( 1987 , Month.JANUARY , 3 ), "CPF3", "name3"));

        crazyDS.addElement(order1);
        crazyDS.addElement(order2);
        crazyDS.addElement(order3);
        System.out.println("---- A: Elements ----");
        crazyDS.printElements();

        System.out.println("---- B: Getting and removing the element with highest priority ----");
        IOrder p1;
```

```
        try {
            p1 = crazyDS.peekElement();
            System.out.println("-selected element");
            p1.printOwner();
            crazyDS.removeElement(p1);
            System.out.println("-elements");
            crazyDS.printElements();
        } catch (CrazyDSException e) {
            e.printStackTrace();
        }

        System.out.println("---- C: Adding an old person ----");
        IOrder order4 = new InternetOrder(new PersonPT(LocalDate.of( 1880 , Month.JUNE , 1 ), "CPF", "name3"));
        crazyDS.addElement(order4);
        crazyDS.printElements();
        System.out.println("-selected element");
        IOrder p2 = crazyDS.peekElement();
        p2.printOwnerFullData();

        System.out.println("---- D: Checking an exception ----");
        try {
            IOrder p3 = crazyDS.getElementAt(1000);
        } catch (CrazyDSException e) {
            //e.printStackTrace();
            System.out.println("-ok: Show error in logs");
        }
    }
}
```

3 Documentação de ajuda

3.1 A classe Period

A classe `java.time.Period` pode ser usada para obter a idade de uma pessoa dado a data de nascimento e a data atual.

```
package com.unicamp.mc322.lab13.help;

import java.time.LocalDate;
import java.time.Month;
import java.time.Period;

public class Main {
    public static void main(String[] args) {
        //How old are you?
        //years
        System.out.println(Period.between(LocalDate.of( 1990 , Month.JUNE , 25 ), LocalDate.now()).getYears());
        //months
        System.out.println(Period.between(LocalDate.of( 1990 , Month.JUNE , 25 ), LocalDate.now()).getMonths());
        //days
        System.out.println(Period.between(LocalDate.of( 1990 , Month.JUNE , 25 ), LocalDate.now()).getDays());
    }
}
```

3.2 Iterable e Iterator

Dica: Para poder usar o *foreach* em Java com a sua classe, a sua classe (estrutura de dados) deve implementar a interface **Iterable**. E, como este precisa de um iterador você deve implementar também o iterador da sua estrutura de dados (p. ex., uma classe *MyCustomDSIterator* que implementa a interface **Iterator**).

Informações acerca destas interfaces podem ser encontradas nos links abaixo:

- `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Iterable.html`
- `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Iterator.html`

Não é necessário usar Iterable e Iterator neste laboratorio, é apenas uma sugestão.