Programação Orientada por Objetos

2º Teste, 15 de junho de 2019 - 10:00

A duração do teste é de 1h40m, mais 20 minutos de tolerância.
O aluno deve permanecer na sala pelo menos 30m.
Responda aos grupos 1-2 e 3-4 em folhas separadas. Identifique todas as folhas.

Grupo 1: (3 Valores)

1.1 (0,5) De acordo com as classes descritas a seguir, selecione **a(s) resposta(s) certas(s)**: (Pode ser mais do que uma!)

```
public class E extends Exception {
  public E (String why) {
    super(why);
  }
}
```

```
public class M {
  public static void main(String[] args) throws E {
    X.m();
  }
}
```

```
public class X {
  public static void m() throw E {
    throws new E ("mensagem");
  }
}
```

a) A classe **E** compila sem erros.
b) A classe **X** compila sem erros.
c) A classe **X** compila com um erro.
d) A classe **X** compila com dois erros.

1.2 (0.5) As exceções constituem um mecanismo de gestão dos erros separado do fluxo de execução normal?
a) Verdadeiro.
b) Falso.

1.3 (0.5 Para testar se duas variáveis referem-se ao mesmo objeto usamos:
a) o método **`equals()`**
b) **`compareTo()`**
c) o operador **`==`**
d) a interface **`Comparable`**

1.4 (0.5) Se pretendemos utilizar um conjunto ordenado, que tipo de coleção devemos usar?
a) **`HashSet`**
b) **`LinkedSet`**
c) **`ArrayList`**
d) **`Array`**

1.5 (0.5) Quando se usam classes de coleção de tipos genéricos, qual dos tipos seguintes não pode ser usado como genérico?
a) **`boolean`**
b) **`Object`**
c) **`String`**
d) **`Integer`**

1.6 (0.5) Indique quais as afirmações **corretas**: (Pode ser mais do que uma!)
a) O método **`Start`** é um método da classe **`Application`**;
b) A classe **`Application`** é super classe da classe **`Stage`**;
c) A classe **`Application`** é a classe que gere os eventos que acontecem na janela da aplicação;
d) O elemento **`root`** da árvore da classe **`Scene`** pode ser um objeto da classe **`Group`**.

Grupo 2: (7 Valores)

A classe **Thread** da Figura 1 representa um conjunto de mensagens (classe **Message**) partilhado por um conjunto de utilizadores (classe **User**). As mensagens são caracterizadas pelo seu identificador (id), texto, autor e lista de tópicos (objetos da classe **String**) também chamados por *tags*.

2.1 (1.0) – Considerando que cada mensagem é identificada de forma unívoca por um ID, complete o construtor da classe **Message**.

2.2 (2.0) – Escreva agora o método **addMessage** da classe **Thread** que permite inserir uma nova mensagem a uma conversa entre utilizadores. A mensagem deverá ser adicionada às mensagens do utilizador, assim como à coleção de mensagens organizada por *tags*. O utilizador também deverá ser adicionado à coleção de utilizadores no caso de não constar dela.

2.3 (1.0) – Atualize agora a classe **User** de forma a que possa ser possível comparar um utilizador com um outro, comparando os respetivos endereços de e-mail. Neste caso, escreva o método da interface **Comparable** que compara um objeto recebido por parâmetro com o objeto da classe **User**.

2.4 (1.0) – Escreva o método **sortedAuthors** da classe **Thread** que permite devolver uma lista ordenada dos utilizadores que publicaram mensagens nessa *thread*.

2.5 (2.0) – Crie agora a método **List<Message> getMessagesByTag(String tag)** que devolve uma lista com todas as mensagens publicadas com a *tag* passada como argumento. Se a *tag* for nula então deve devolver todas as mensagens. A lista devolvida não deve conter repetições das mensagens.

Grupo 3: (4 Valores)

Continuando o programa do grupo anterior pretende-se agora fazer algumas alterações na estrutura. Sendo assim, considere agora os tipos **TextIssue** e **StringArgumentException** mostrados na Figura 2.

3.1 (2.0) – No código da aplicação podem existir vários problemas se for criado um utilizador com um email inválido. Neste caso, para evitar este problema e testar a solução faça o seguinte:

- Reescreva o construtor da classe **User** para que seja lançada uma exceção do tipo **StringArgumentException** sempre que o email venha com **null**, sem texto (ou apenas espaços em branco) ou não contenha o caracter "**@**". Deve utilizar o tipo **TextIssue** de acordo o erro ocorrido.
- Exemplifique a captura da exceção gerada com a criação de um novo utilizador. Tenha em atenção que no caso de existir uma exceção deve ser mostrada uma mensagem de erro na consola como no exemplo abaixo:

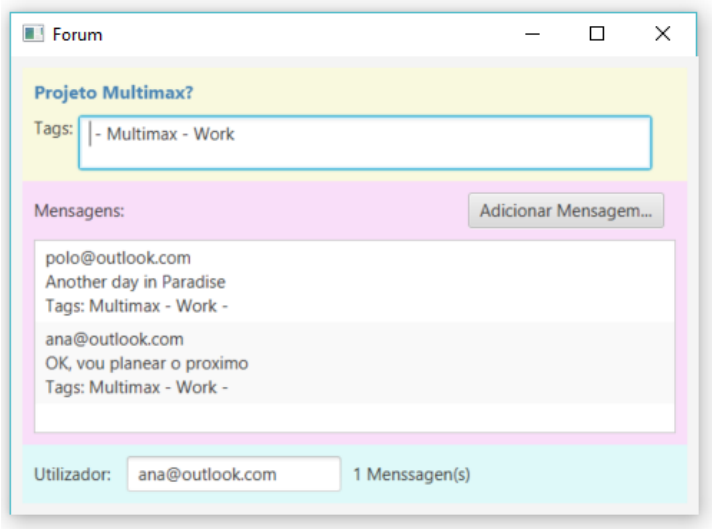
```
Erro: email
Causa: sem texto
```

Nota: as duas linhas de texto foram obtidas com uma instrução **System.out.println** diferente para cada linha.

3.2 (2.0) – Verificou-se no código mostrado algumas situações em que existia uma relação de um para muitos entre objetos. Por exemplo, para um utilizador existiam várias mensagens, ou para uma mensagem estavam associados vários *tags*. Sendo assim, defina o seguinte código:

- Crie o tipo genérico **OneToMany** que associa uma chave a uma lista de valores. Implemente o método construtor que recebe como argumento apenas o objeto chave e cria a lista de valores a associar a esse objeto. Defina ainda o método que permite adicionar um valor à lista de valores, o método que retorna a lista de valores e o método que retorna o objeto chave.
- Exemplifique a utilização desta classe genérica com a substituição dos atributos **text** e **tags** da classe **Message** por um objeto da classe criada. Reescreva apenas o construtor desta classe utilizando o novo atributo e também o método **getTags** (sem alterar a sua assinatura). Nota: para além dos dois métodos pedidos, basta apenas escrever o novo atributo, não necessita de escrever o resto do código da classe.

Grupo 4: (6 Valores)



Pretende-se desenvolver uma aplicação **JavaFX** que mostre as mensagens de um fórum como está representado na figura de cima. A classe desta aplicação – **`MessengerFX`** encontra-se na Figura 3. A cena mostrada na janela da aplicação tem como raiz um objeto da classe **`ForumPane`** que aparece na Figura 4.
Nota: Foram retirados das Figuras 3 e 4 todas as instruções de formatação dos controlos e de espaçamentos envolvidos e não são necessários no código que é pedido neste grupo.

4.1 (3.0) – Sabendo que no construtor da classe **`ForumPane`** são definidos os painéis que aparecem no topo, centro e base, cujo código é mostrado na Figura 4 crie o seguinte código:

- Escreva o código do construtor que cria os controlos mostrados no painel **`threadPane`**. A informação mostrada neste painel corresponde ao nome e tópicos da *thread* e é obtida através do objeto **`thread`** recebido. No final o painel com os controlos contendo a visualização da informação deve ser colocado no topo do **`ForumPane`**.

- Escreva o código do construtor que cria os controlos mostrados no painel **`userPane`**. A informação mostrada neste painel corresponde à lista de mensagens da **`thread`**. Use o método **`getMessagesByTag(null)`** para obter a lista de todas as mensagens que vão ser mostradas no controlo **`ListView`**. No final o painel com os controlos contendo a visualização da informação deve ser colocado no centro do **`ForumPane`**.

- Escreva o código do construtor que cria os controlos mostrados no painel **`messagesPane`**. A informação mostrada neste painel corresponde ao email e número de mensagens do utilizador ligado (**`loggedUser`**). A única ação do botão "Adicionar Mensagem" é a chamada ao método **`addMessage()`** definido na classe. No final o painel com os controlos contendo a visualização da informação deve ser colocado na base do **`ForumPane`**.

4.2 (2.0) – O método **`addMessage()`** da classe **`ForumPane`** é responsável por criar a janela de diálogo onde o utilizador escreve a mensagem e associa-lhe um conjunto de tópicos. Para o efeito é criado um objeto da classe **`MessageDialog`** que se mostra na Figura 5. Tendo em conta o código desta classe faça um esboço da janela que é mostrada incluindo todos os detalhes relevantes.

4.3 (1.0) – Crie agora o método **`addMessage()`** da classe **`ForumPane`**. Este método cria e mostra a janela de diálogo (um objeto da classe **`MessageDialog`**). Para se obter a mensagem adicionada pelo utilizador através desta janela pode-se usar o método **`getMessage()`**. Este método retorna **`null`** se não foi adicionada nenhuma mensagem ou se o utilizador saiu com *Cancel*, caso contrário retorna a nova mensagem.

```
public class Message {
   private static int messageNumber = 0;
   private int id;
   private String text;
   private List<String> tags;  // tópicos
   private User author;

   public Message (User author, String text,
                   String[] tags) {
      // Alínea 2.1
   }

   public User getAuthor() {
      return author;
   }

   public List<String> getTags() {
      return tags;
   }

   public int getId() {
      return id;
   }

   public String toString() {
      String message = author.getEmail();
      message +="\n"+this.text+"\n"+"Tags: ";
      for (String tag: tags){
         message += tag + " - ";
      }
      return message;
   }
}
```

```
public class User implements Comparable{
   private String email;
   private String password;
   private List<Message> messages;

   public User (String email,
                String password) {
      this.email = email;
      this.password = password;
      this.messages = new ArrayList<>();
   }

   public String getEmail() { return email; }

   public int getNumberOfPosts() {
      return messages.size();
   }

   public void addMessage(Message message) {
      messages.add(message);
   }

   @Override
   public String toString(){
      String lista = "\n-----------\n";
      lista += "mensagens de "+email+":\n";
      for(Message m : messages){
         lista += m.toString();
      }
      return lista;
   }

   // Alínea 2.3 – implementar "Comparable"

   public int hashCode() {/*Código omitido*/}

   public boolean equals(Object obj)
        {/*Código omitido*/}
}
```

```
public class Thread {
    private String name;
    private Map<String, User> users; // email, user
    private Map<String, List<Message>> messagesByTag;
                                      // tag, lista mensagens

    public Thread(String name) {
        this.name = name;
        users = new HashMap<>();
        messagesByTag = new HashMap<>();
    }

    public void addMessage(User user, Message message) {
       // Alínea 2.2
    }

    public ArrayList<User> sortedAuthors() {
       // Alínea 2.4
    }

    public List<Message> getMessagesByTag(String tag) {
       // Alínea 2.5
    }

    public List<String> getTags() {
        // Devolve a lista ordenada das tags – código omitido
    }
}
```

```
public class Messenger {
   public static void main(String[] args) {
      User user1 = new User("polo@outlook.com","lol");
      User user2 = new User("ana@outlook.com","xpto");
      String[] tagSet = {"Multimax","Work"};
      Message message1 = new Message(user1,
                         "Another day in Paradise",tagSet);
      Message message2 = new Message(user1,
                         "Acabamos o sprint amanha",tagSet);
      Message message3 = new Message(user2,
                         "OK, vou planear o proximo",tagSet);
      Thread project1 = new Thread("Projeto Multimax?");
      project1.addMessage(user1,message1);
      project1.addMessage(user1,message2);
      project1.addMessage(user2,message3);
      for(User u : project1.sortedAuthors()){
         System.out.println(u);
      }

System.out.println(project1.mainPostersFromTag("Work"));
    }
}
```

```
// output: exemplo em modo consola
-----------
mensagens de ana@outlook.com:
ana@outlook.com
OK, vou planear o proximo
Tags: Multimax - Work -

-----------
mensagens de polo@outlook.com:
polo@outlook.com
Another day in Paradise
Tags: Multimax - Work - polo@outlook.com
Acabamos o sprint amanha
Tags: Multimax - Work -

Os mais participativos:
polo@outlook.com: 2
ana@outlook.com: 1
```

Figura 1: Classes **Message**, **Messenger**, **Thread** e **User**

```
public enum TextIssue {
    NULL, BLANK, INVALID;

    @Override
    public String toString() {
        switch (this) {
            case NULL:
                return "nulo";
            case BLANK:
                return "sem texto";
            case INVALID:
                return "inválido";
        }
        return super.toString();
    }
}
```

```
public class StringArgumentException extends IllegalArgumentException{

    private TextIssue textIssue;

    public StringArgumentException(String string, TextIssue textIssue) {
        super(string);
        this.textIssue = textIssue;
    }

    public TextIssue getTextIssue() {
        return textIssue;
    }
}
```

Figura 2: Tipo enumerado **TextIssue** e classe de exceção **StringArgumentException**

```
public class MessengerFX extends Application {

    public void start(Stage primaryStage) {

        User user1 = new User("polo@outlook.com","lol");
        User user2 = new User("ana@outlook.com","xpto");
        String[] tagSet = {"Multimax","Work"};
        Message message1 = new Message(user1,"Another day in Paradise",tagSet);
        Message message2 = new Message(user1,"Acabamos o sprint amanha",tagSet);
        Message message3 = new Message(user2,"OK, vou planear o proximo",tagSet);

        Thread t1 = new Thread("Projeto Multimax?");
        t1.addMessage(message1);
        t1.addMessage(message2);
        t1.addMessage(message3);

        ForumPane root = new ForumPane(t1);

        Scene scene = new Scene(root, 600, 400);

        primaryStage.setTitle("Forum");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

Figura 3: classe **MessengerFX**

```
public class ForumPane extends BorderPane {

    private User loggedUser;
    private Thread thread;
    private ObservableList<Message> messageList;

    public ForumPane(Thread thread) {

        if (thread == null) { throw new IllegalArgumentException("Thread não existente"); }
        this.thread = thread;
        loggedUser = thread.sortedAuthors().get(0);  // para teste

        // Painel do topo - mostra a lista de tags e o nome da Thread
        VBox threadPane = new VBox(10);

        // Painel do centro - mostra a lista de mensagens e inclui um botão
        // que permite ao utilizador ligado adicionar uma nova mensagem
        VBox messagesPane = new VBox(10);

        // Painel da base - mostra o utilizador ligado e o total de mensagens enviado por ele
        HBox userPane = new HBox(10.0);
        userPane.setAlignment(Pos.CENTER_LEFT);

        // Alínea 4.1 – ThreadPane

        // Alínea 4.1 – userPane

        // Alínea 4.1 - messagesPane
    }

    private void addMessage() {
        // Alínea 4.3
    }
```

Figura 4: Classe **ForumPane**

```
public class MessageDialog extends Stage {
   private Message message;
   private User user;

   public MessageDialog(User user) {
      this.user = user;
      message = new Message(user, "", new String[0]);
      setTitle("Adicionar Mensagem");

      HBox topPane = new HBox();
      topPane.setAlignment(Pos.CENTER);

      Text textMessage = new Text("Mensagem nº ");
      textMessage.setFill(Color.BLACK);
      textMessage.setFont(new Font(20));

      Text text = new Text("" + message.getId());
      text.setStroke(Color.BLACK);

      Rectangle rectangle =
                 new Rectangle(50, 40, Color.AQUA);
      rectangle.setStroke(Color.BLACK);

      StackPane painel = new StackPane();
      painel.getChildren().addAll(rectangle, text);

      topPane.getChildren().addAll(textMessage, painel);

      GridPane gp = new GridPane();
      gp.setHgap(10);
      gp.setVgap(20);

      Label lblMessage = new Label("Mensagem: ");
      TextArea txtMessage = new TextArea();
      txtMessage.setWrapText(true);
      txtMessage.setMaxSize(200, 40);

      gp.add(lblMessage, 0, 0);
      gp.add(txtMessage, 1, 0);

      for (int i = 1; i < 4; i++) {
         gp.add(new Label("tag" + i), 0, i);
         gp.add(new TextField(), 1, i);
      }

      VBox vBoxMain = new VBox(10);
      vBoxMain.setPadding(new Insets(10.0));

      Button okButton = new Button("Ok");
      Button cancelButton = new Button("Cancel");
      HBox buttonsHBox = new HBox(30);
      buttonsHBox.setAlignment(Pos.CENTER_RIGHT);
      buttonsHBox.getChildren().addAll(okButton,
                                       cancelButton);
      buttonsHBox.setPadding(new Insets(20));
      vBoxMain.getChildren().addAll(topPane, gp,
                                    buttonsHBox);

      setResizable(false);
      initStyle(StageStyle.UTILITY);
      initModality(Modality.APPLICATION_MODAL);
      setIconified(false);
      centerOnScreen();
      setScene(new Scene(vBoxMain, 350, 400));

      // Restante Código omitido
   }

    public Message getMessage() {
       // Código omitido
    }
}
```

Figura 5: Classe **MessageDialog**