

Índice

Quizzes	2
Quiz 1 – Polimorfismo	2
Quiz 2 – Classes Abstratas e Interfaces	3
Quiz 3 – Desenvolvimento de aplicações	6
Quiz 4 – Genéricos e Coleções	9
Quiz 5 – Exceções e Input/Output	13
Quiz 6 – Introdução à JAVAFX	17
Quiz 7 – JAVAFX – controlo e eventos	21
Quiz 8 – JAVAFX – Propriedades e controlos avançados	23
Resumindo (Sumários dos slides)	26
Classes abstratas e Interfaces	26
Exceções	26
Introdução ao JAVAFX	27
JAVAFX – Eventos e Painéis	28
JAVAFX – Controlos	29
JAVAFX – Propriedades	29
JAVAFX – Controlos II	30
JAVAFX – Janelas e Formas	31

Quizzes

Quiz 1 – Polimorfismo

1 – O polimorfismo é o ato de usar o mesmo nome de **método** para indicar implementações diferentes para métodos diferentes para métodos baseados no tipo de **objeto**.

2 – Quando um método de subclasse tem o mesmo nome e tipos de argumento que um método de superclasse, o método da subclasse **substitui** o método de superclasse.

3 – *Method lookup* é uma tarefa realizada em **tempo de execução** e consiste em determinar **qual método deve ser executado**.

4 – Considere as seguintes classes:

```
class Animal {
    public String makeSound() {
        return "não produz som"; }
}

class Cat extends Animal {
    public String makeSound() {
        return "Miau"; }
}

class StreetCat extends Cat {
}

class AmericanBobtail extends Cat {
    public String makeSound() {
        return "Miauuu"; }
}
```

Considere agora o seguinte código no programa principal:

```
ArrayList<Cat> cats = new ArrayList<>();

cats.add(new Cat());
cats.add(new StreetCat());
cats.add(new AmericanBobtail());

for( Cat c : cats )
    System.out.println( c.makeSound());
```

O que é mostrado no ecrã quando o programa principal é corrido?

Resposta:

Miau

Miau

Miauuu

5 – Considerando que a classe *Ticket* é uma generalização da classe *PlaneTicket*, e que a classe *Ticket* possui um método *show*.

O seguinte conjunto de instruções:

```
PlaneTicket ticket = new Ticket("Paris-Lisboa","TAP345");  
ticket.show();
```

Está:

- **Incorreta, pois não é possível atribuir a uma variável de uma dada classe uma instância de uma superclasse da mesma.**

Quiz 2 – Classes Abstratas e Interfaces

1 – Considere o seguinte código em JAVA

```
public interface Washable  
{ //código }  
  
public abstract class Vehicle implements Washable  
{ //código }  
  
public class Bike extends Vehicle  
{ //código }  
  
public class Car extends Vehicle  
{ //código }
```

Qual das seguintes afirmações está **correta**:

- **Car implementa Washable.**

2 – A palavra reservada *Abstract* é usada para definir:

- **Classe abstratas e métodos abstratos**

3 – Considere o seguinte código em JAVA

```
public interface Descriptible {  
    public String getDescription();  
}
```

E assumo que a classe *Book* implementa a interface *Descriptible*.

Identifique a afirmação correta:

- A classe *Book* se não for abstrata e não herdar de nenhuma outra classe tem que obrigatoriamente implementar o método *getDescription*

4 – Considerando o seguinte código:

```
interface Chargeable { int charge(); }  
  
class Device implements Chargeable {  
    public int charge() { ... }  
    public void turnOn() { ... }  
    ... }  
  
Chargeable computer = new Device();
```

Qual das seguintes linhas está **correta**:

- `System.out.println(computer.charge())`

5 - Um método abstrato é normalmente usado para:

- Declarar um método que deve ser implementado nas classes derivadas

6 – Considere que existem as interfaces *Descriptible* e *Colorable*

Qual das seguintes instruções em JAVA se pode considerar **correta**:

- `Public class Cubo implements Descriptible{...`

7 – Considere o seguinte código em JAVA

```
public interface A {  
    public int metodoA (int valor);  
}  
  
public interface B {  
    public String metodoB (String valor);  
}  
  
public interface C extends A, B  
    { //código }
```

Qual das seguintes afirmações está correta.

- Os métodos *metodoA* e *metodoB* são herdados pela interface C.

8 – Qual o mecanismo que se deve utilizar quando se pretende que as classes duma dada hierarquia implementem um determinado comportamento?

- Classes abstratas

9 – Considerando que a classe *Car* é abstrata, qual dos seguintes conjuntos de instruções está **correto**:

- `Car[] listC = new Car[2]`

10 – Qual a afirmação **correta**?

- Um método abstrato não possui código.

11 – Considere o seguinte código em JAVA

```
public interface Cleaner{public void clean();}
```

E assumo que a classe *Vacuumcleaner* implementa a interface *Cleaner*.

Qual dos seguintes conjuntos de instruções é **correto**:

- `Cleaner c = new Vacuumcleaner(); c.clean();`

Quiz 3 – Desenvolvimento de aplicações

1 – Classes e Métodos pouco coesos...

- **Difícultam a expansão de funcionalidades;**
- **Difícultam a correção de problemas.**

2 – A Coesão é uma medida da quantidade e diversidade dos assuntos (ou tarefas) pelos quais uma entidade é responsável

- **Verdadeiro**

3 – Quais são as etapas a seguir para elaborar cartas CRC

- **Etapla 1 – Encontrar as classes;**
- **Etapla 2 – Encontrar as Responsabilidades;**
- **Etapla 3 – Definir os colaboradores de cada classe;**
- **Etapla 4 – Organizar as cartas para mostrar aos programadores.**

4 – Considere as seguintes classes:

```
1  public class Calculator
2  {
3      private int number1;
4      private int number2;
5      // ...
6      public int input() { ... }
7      public int add () { ... }
8      public int divide() { ... }
9      public int multiply() { ... }
10     public void displayResult(int result) { ... }
11     public void displayError(int result) { ... }
12 }
```

Existem comportamentos que não são válidos para todos os objetos da classe. Que alteração deve ser feita:

- **Criar uma nova classe (*Display*), e associar à classe *Calculator* o método *multiply* e o método *add* e à classe *Display* os métodos *displayResult* e *displayError*.**

5 – Consideres as seguintes classes:

```
1 public class Person {
2     private String name,id;
3
4     public Person(String name, String id) {
5         this.name = name;
6         this.id = id;
7     } }
8
9 public class Pet {
10     private String name;
11     private Person owner;
12     private Address ownerAddress;
13     private int age;
14
15     public Pet(String name, int age) {
16         this.name = name;
17         this.age = age;
18         owner=null;}
19
20     public void setOwnerName(String name) {
21         owner.setName(name);}
22
23     public int getAge() {
24         return age;}
25
26     public void incrementAge() {
27         age++;}
28
29     public String getName() {
30         return nome;}
31
32     public void setName(String name) {
33         this.name = name;}
34 }
```

Qual dos seguintes problemas acha que deverá existir?

- **Alto acoplamento de identidades**

6 – Considere a seguinte classe:

```
1 public class BoardGame {  
2     private Piece[] pieces;  
3     private Board board;  
4     private Room gameRoom;  
5     private Date gameDate;  
6  
7     // métodos...  
8  
9 }
```

- **Mistura de domínios**

7 – Considere a existência da classe Veículo e Condutor.

Se considerarmos que Condutor herda de Veículo...

- ☐ a. Todas as outras respostas estão incorretas
- ☐ b. está correcto, pois estamos a permitir a reutilização de código através do uso da herança.
- ☒ c. é um erro pois estamos a criar acoplamento de identidades entre as duas classes ✖
- ☐ d. é um erro pois estamos a criar acoplamento de representação entre classes

- **Todas as outras respostas estão incorretas**

8 – Pediu-se a um aluno para construir um programa para uma agência de viagens. O aluno identificou uma classe *TravelKit* onde iria incluir informações detalhadas sobre o cliente (nome, morada, telefone, nº passaporte). Se o modelo proposto pelo aluno fosse implementado que problemas encontrava?

- **Uma baixa coesão**

Quiz 4 – Genéricos e Coleções

1 – O ciclo **Foreach** é um modo de iteração compacto que usa um iterador implicitamente, simples e que funciona com Arrays.

- **Verdadeiro**

2 – É valido o seguinte código?

```
<HashMap<Integer,String> aMap = new Map<>();
```

- **Falso**

3 – Considere o seguinte código em JAVA:

```
HashSet<String> set = new HashSet<String>();  
  
set.add("IP");  
  
set.add("POO");  
  
set._____;
```

Assinale TODAS as opções possíveis para completar a última instrução de forma a este excerto de código poder ser executado sem dar nenhuma exceção, ou erro de compilação

- `clear();`
- `remove("IP");`
- `size();`
- `get("POO").`

4 – Um iterador (iterator) serve para operar sobre os elementos que estão dentro de uma coleção um a um. Indique o que falta nos espaços assinalados com _____, respectivamente:

```
LinkedList<Integer> lista = new LinkedList<Integer>();  
Iterator<Integer> it = lista.iterator();  
while(_____) {  
    Integer a = _____;  
    System.out.println(a);  
}
```

- `it.hasNext()` e `it.next()`

5 – A interface `List<E>` está na hierarquia da interface `Collection<E>`.

Assim todas as coleções que implementam essa interface possuem os métodos:

- `contains()`, `set()`, `add()`, `get()`.

6 – Considere o seguinte problema:

Temos uma lista com os cromos que o João tem (`tradingCardsList`). Queremos calcular qual o número de cromos que o João tem para a troca (os que são repetidos). Uma forma possível de implementarmos em JAVA o cálculo do número de cromos para a troca é:

1. Definirmos uma variável *uniqueTradingCards* como sendo do tipo `HashSet<TradingCard>`;
2. Adicionarmos cada um dos cromos da lista inicial no `HashSet` criado.
3. Calcularmos a diferença entre o tamanho da lista inicial e da coleção *uniqueTradingCards*.

7 – Considere o seguinte código

```
HashMap<Integer, Invoice> invoices = new HashMap<>();  
  
invoices.put(1, new Invoice(1) );  
invoices.put(2, new Invoice(2) );
```

Sabendo que o número inteiro representa o número da fatura, qual dos seguintes ciclos escolheria se quisesse iterar todas as faturas existentes para obter a informação de cada uma:

- **for(Invoice iv : invoices.values())**

8 – Indique a sequência de passos para realizar as seguintes ações em JAVA:

(1) Criar uma instância da classe `HashMap` denominada `catalog`, onde a chave é do tipo `String` e o valor do tipo `Book`.
(2) Adicionar o livro l1, ao catalogo.
(3) Aceder ao livro com a chave "A0005".
(4) Remover o elemento com a chave "A0005"
(5) Imprimir no ecrã o número de livros do catalogo.

1. `HashMap<String,Book> catalog=new HashMap<String,Book>()`
2. `catalog.put("A0005",l1)`
3. `catalog.get("A0005")`
4. `catalog.remove("A0005")`
5. `System.out.println(catalog.size())`

9 - O que é genericidade?

- Um conceito que permite não especificar um tipo preciso para uma classe, coleção ou método, a fim de ter código reutilizável.

10 - Quais das afirmações que aparecem a seguir são verdadeiras quando se comparam dois objetos da mesma classe sabendo que os métodos `hashCode` e `equals` foram implementados corretamente? (Escolha todas as respostas que se aplicam)

- Se o método `equals()` retornar `true`, os métodos `hashCode()` devem retornar o mesmo valor.,
- Se os métodos `hashCode()` retornarem o mesmo valor, a comparação usando o método `equals()` pode ou não retornar `true`.

11 – Considere o seguinte código:

```
List list = new ArrayList();  
list.add(10);  
list.add("10");  
  
List<Integer> list = new ArrayList<Integer>();  
list.add(10);
```

Da lista de vantagens associadas à utilização de tipos genéricos apresentada abaixo, qual das vantagens é ilustrada pelo código.

Selecione uma opção:

- **Segurança de tipo**: podemos armazenar apenas um único tipo de objetos nos genéricos. Não permite armazenar outros objetos.

12 – Considere o código a seguir:

```
class MyGen<T>{  
    T obj;  
    void add(T obj){this.obj=obj;}  
    T get(){  
        return obj;}  
}  
class App{  
    public static void main(String args[]){  
        MyGen<____> m=new MyGen<____>();  
        m.add(2);  
        System.out.println(m.get());  
    }  
}}
```

De acordo com a genericidade, que classe deverá constar no código nas zonas por completar, assinaladas com ____:

- **Integer**

Quiz 5 – Exceções e Input/Output

1 - Qual é o nome da classe abstracta base para canais (streams) que lidam com entrada de caracteres?

- **Reader**

2 - Indique a ordem correta de executar as instruções para escrever no Ficheiro hoje.txt uma frase no programa abaixo

```
public class Teste{  
    public static void main (String arg[]) {  
        //Criar acesso ao ficheiro  
  
        <instrucao 1>  
  
        <instrucao 2>  
  
        <instrucao 3>  
  
        // Escrever frase  
  
        <instrucao 4>  
  
        print_writer.flush();  
  
    }  
}
```

1. **FileWriter file_writer = new FileWriter (new File("hoje.txt"))**
2. **Writer buf_writer = new BufferedWriter (file_writer)**
3. **PrintWriter print_writer = new PrintWriter (buf_writer)**
4. **print_writer.println ("Hoje esta sol")**

3 - Qual das seguintes exceções capturaria no tratamento de um erro provocado por uma divisão por zero?

- **ArithmeticException**

4 – Indique a ordem correta de executar as instruções para ler do Ficheiro Bomdia.txt uma frase

```
public class Teste {  
  
    public static void main (String arg[]) {  
  
        <instrucao 1>  
  
        try{  
            <instrucao 2>  
            <instrucao 3>  
            System.out.println ("String lida: " + primeiraLinha);  
        }  
  
        <instrucao 4> {  
            System.out.println ("Ficheiro não encontrado!");  
            System.exit (0);  
  
        }  
    }  
}
```

1. `File ficheiro = new File ("Bomdia.txt")`
2. `Scanner sc = new Scanner(ficheiro)`
3. `String primeiraLinha = scanner.nextLine()`
4. `catch (FileNotFoundException e)`

5 – Que método da classe *File* é utilizado para testar a existência de um ficheiro ou diretoria

- `exists()`

6 - A diferença entre exceções verificadas e não-verificadas é que:

- as exceções verificadas derivam da classe `Exception` e as não-verificadas da classe `RuntimeException`

7 - Tendo em conta o seguinte código de um método:

```
void method()
{
    int a=-12, b=3, res;

    try {
        res = sumPositives( a, b );
    }
    catch(ErrorSumException e)
    {
        System.out.println("Erro: " + e.getMessage() + e.getValor());
        // Resultado no ecrã: "Erro: Valor negativo: -12"
    }
} // metodo
```

Se tivesse de implementar a classe `ErrorSumException` qual considera que seria o construtor mais adequado dos que se mostram a seguir:

Selecione uma opção:

Resposta (anexo):

```
public ErrorSumException( String message, int valor)
{
    super(message);
    this.valor = valor;
}
```

8 - Dado o seguinte excerto de código, indique as afirmações corretas (podem ser várias)

```
public class Main {  
  
    public static void main(String[] args){  
        Data x = new Data (2012, 15, 31);  
        Dia d = -1;  
  
        try{  
            System.out.print("Bingo 1 &");  
            x.setDia(32);  
            x.troca(d, -5);  
  
        }  
        catch (ValorInvalidoException e){  
            System.out.print("Bingo 0");  
        }  
  
        catch (TrocaInvalidaException e){  
            System.out.println("  & Bingo 2");  
        }  
    }  
}
```

Selecione uma ou mais opções:

- Após a execução deste programa é possível que apareça escrito no ecrã "Bingo1 & & Bingo2"
- Podemos afirmar que na execução deste programa aparecerá SEMPRE no ecrã escrita a palavra "Bingo 1 &"

9 - Desserializar um objeto pode causar uma `ClassNotFoundException`

- **Verdadeiro**

10 – Uma exceção é um evento gerado pelo código do programa, portanto pode ser capturado pelo compilador

- **Falso**

11 - Qual é o método que existe na classe `Scanner` que pode ser usado para ler um valor inteiro entre os que são mostrados?

- **`nextShort()`**

Quiz 6 – Introdução à JAVAFX

1 – Considere o seguinte código de uma aplicação em JavaFX e explique o seu funcionamento:

```
1  @Override
2  public void start(Stage primaryStage) {
3
4      primaryStage.setTitle("Slide 9 Text Fonts");           // <1>
5      Group root = new Group();
6      Scene scene = new Scene(root, 550, 250);
7
8      Text text5 = new Text(50, 50, "Font Monospaced BLACK"); // <2>
9      Font monoFont = Font.font("Monospaced", 40);           // <3>
10     text5.setFont(monoFont);
11     text5.setFill(Color.BLACK);                             // <4>
12     root.getChildren().add(text5);
13
14     primaryStage.setScene(scene);
15     primaryStage.show();
16 }
```

1. Mostra o texto "Slide 9 Text Fonts" na barra de título da janela da aplicação
2. Cria um texto com a string "Font Monospaced BLACK" que deverá ser mostrada na posição X=50, Y=50
3. Cria uma fonte mono espaçada e com 40 pontos de tamanho
4. Altera a cor das letras do texto para preto

2 – Considere a interface do utilizador da figura 1. Faça a correspondência entre as letras A até G da figura 2 para identificar as classes da estrutura da aplicação:

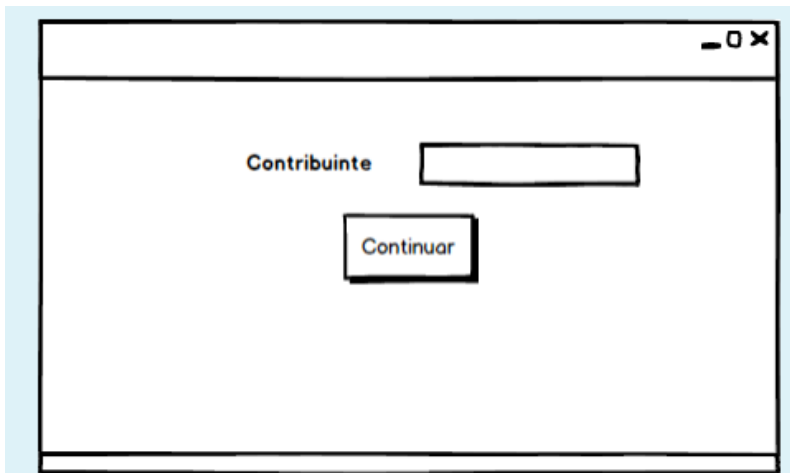


Figura 1

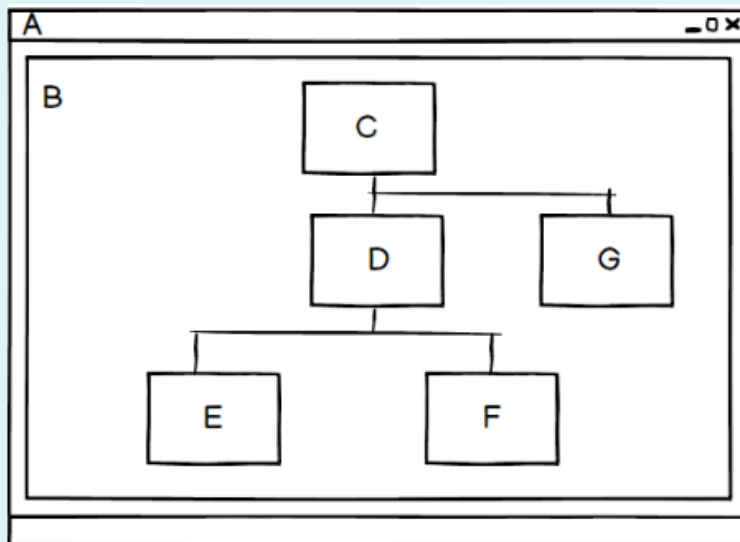


Figura 2

- A. Stage
- B. Scene
- C. StackPane
- D. HBox
- E. Label
- F. TextField
- G. Button

3 – Quais das seguintes classes são nós (ou seja, derivadas da classe *Node*)?

- Rectangle;
- Text;
- Group;
- Shape.

4 – Complete o seguinte código:

```
{  
  
    // ...  
  
    Group circles = new Group();  
  
    for (int i = 0; i < 30; i++) {  
  
        Circle circle = new Circle(150+2*i, Color.web("white", 0.05));  
        circle.setStrokeType(StrokeType.OUTSIDE);  
        circle.setStroke(Color.web("white", 0.16));  
        circle.setStrokeWidth(4);  
  
        //<< escolha a linha de código >>  
  
    }  
}
```

- circles.getChildren().add(circle);

5 - Pretende-se criar uma aplicação gráfica que mostre a seguinte frase escrita a azul - " BOM DIA!!".

Indique a ordem correcta de instruções a implementar na classe abaixo.

```
1 public class JavaFXTesteMoodle extends Application {
2
3 public static void main(String[] args) {
4     launch(args);
5 }
6
7 @Override
8 public void start(Stage primaryStage) {
9
10     // A - Criar o texto a azul
11     // B - Definir um contentor do tipo StackPane*
12     // C - Adicionar o texto*
13     // D - Criar uma cena, associada ao painel criado
14
15     primaryStage.setScene(scene);
16     primaryStage.show();
17 }
18
19 }
```

- A. Criar o texto a azul → `Text t = new Text (20, 20, "BOM DIA"); t.setFill(Color.BLUE);`
- B. Definir um contentor do tipo StackPane → `StackPane root = new StackPane();`
- C. Adicionar o texto → `root.getChildren().add(t);`
- D. Criar uma cena, associada ao painel criado → `Scene scene = new Scene(root, 60,60);`

6 – Em JavaFX:

A classe que contém o método main é subclasse de Application.

No método main() lançamos a aplicação passando os argumentos para o método

<< complete aqui >>

Quando a aplicação tiver sido inicializada a infraestrutura do javaFX invocará o método `Application.start()`

- **`Application.launch()`**

7 - Indique quais as afirmações erradas sobre a classe `Application` do JavaFx.

- *Application* é a classe que gere os eventos que acontecem na janela da aplicação.,
- A classe *Application* é a superclasse da classe *Stage*.

Quiz 7 – JavaFX – controlo e eventos

1 - Se pretendemos que um conjunto de 6 nós, apareçam nas células de numa tabela com 6 linhas e 1 coluna devemos usar um contentor do tipo:

- **VBox**

2 - Faça a correspondência correta entre o evento que se pretende capturar e o método a utilizar.

- Detetar quando uma tecla do teclado é premida -> **setOnKeyPressed**
- Detetar quando o mouse sai da área do nó -> **setOnMouseExited**
- Detetar quando uma ação de arrastar com o mouse é iniciada -> **setOnDragDetected**

3 - Em JavaFX qual o nome do pacote onde se encontram os componentes gráficos usados na interface com o utilizador?

- `javafx.scene.control`

4 - Se pretendemos que um conjunto de nós, apareçam alinhados **verticalmente** devemos usar um contentor do tipo:

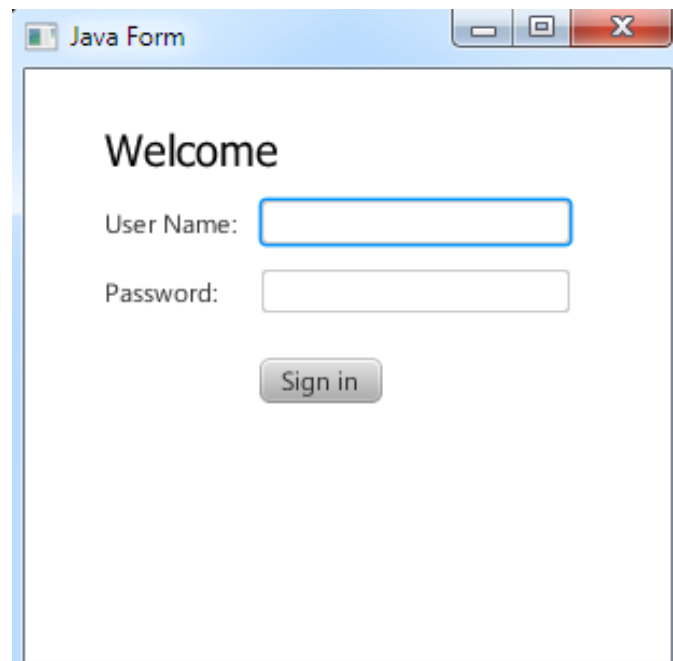
- **VBox**

5 -

```
Rectangle rectangle = RectangleBuilder << 1 >>  
.x(50)  
<< 2 >> //desvio vertical  
<< 3 >> //largura  
.height(70)  
<< 4 >> ;
```

1. `.create();`
2. `.y(100);`
3. `.width(100);`
4. `.build();`

6 - Pretende-se fazer uma aplicação com o formulário seguinte:



The image shows a Java Swing window titled "Java Form". Inside the window, there is a login form. At the top, the word "Welcome" is displayed in a large, bold, black font. Below it, there are two labels: "User Name:" and "Password:". Each label is followed by a text input field. The "User Name:" input field has a blue border, while the "Password:" input field has a gray border. Below these input fields is a "Sign in" button with a gray background and black text. The window has a standard Mac OS X-style title bar with a red close button, a yellow maximize button, and a green minimize button.

Escolha o painel mais apropriado para conseguir a formatação pretendida.

- **GridPane**

Quiz 8 – JavaFX – Propriedades e controlos avançados

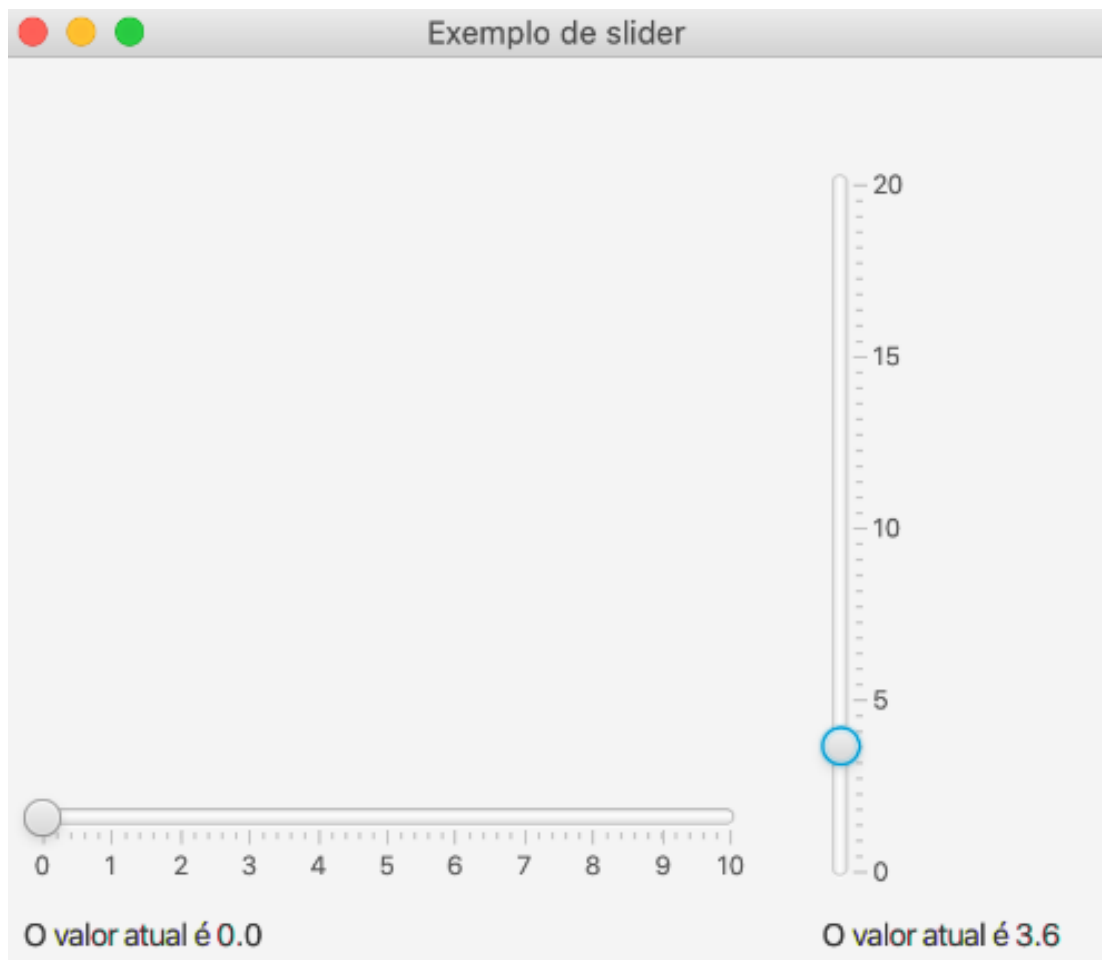
1 - A classe `ComboBox` pode conter diversos tipos de itens, por exemplos imagens.

- **Verdadeiro**

2 - Se consideramos a classe `TextField` do JavaFX, os métodos `setText` e `getText` deste controlo permitem aceder

- **as propriedades**

3 - Considere o programa composto pela janela abaixo com dois controlos deslizantes *Slider*.



Considere o extrato de código a seguir que configura o controlo deslizante vertical:

```
Slider vertSlider = new Slider(0, 20, 0);  
vertSlider.setMinHeight(vertSliderHeight);  
vertSlider.setShowTickMarks(true);  
vertSlider.setShowTickLabels(true);  
vertSlider.setSnapToTicks(true);  
vertSlider.setMajorTickUnit(5.0);  
vertSlider.setMinorTickCount(10);  
// <instrução>
```

Qual é a instrução em falta para finalizar a configuração das propriedades?

Selecione uma opção:

- **vertSlider.setOrientation(Orientation.VERTICAL);**

4 - Em JavaFX uma *CheckBox* pode estar num estado indeterminado, o que significa que não está seleccionado nem não está seleccionado.

- **Verdadeiro**

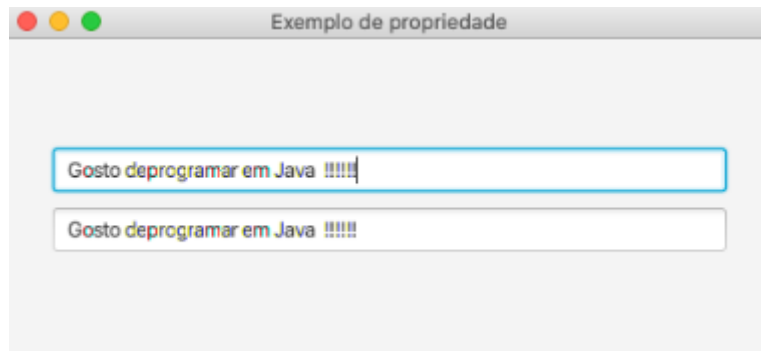
5 – Considere as duas instruções a seguir.

```
final ObservableList<String> students = -----.observableArrayList();  
final ListView<String> studentsListView = new ListView<>(students);
```

Indique que instrução deve ser usada no lugar do símbolo -----

- **FXCollections**

6 - Considere o programa composto pela janela abaixo que permite que o texto inserido pelo utilizador no *TextField* de cima aparece em simultâneo no controlo de baixo.



Considere, também, o extrato de código abaixo

```
public void start(Stage stage)
{
    TextField top = new TextField();
    TextField bottom = new TextField();

    // vincula a propriedade de texto do campo de texto inferior à propriedade do campo do topo
    // <instrução>

    VBox root = new VBox(10);
    root.getChildren().addAll(top, bottom);
    root.setAlignment(Pos.CENTER);

    Scene scene = new Scene(root, 480, 200);
    stage.setScene(scene); stage.setTitle("Exemplo de propriedade");
    stage.show();
}
```

Qual deverá ser a <instrução> a seguir aos comentários, que permite vincular as propriedades?

- **bottom.textProperty().bind(top.textProperty());**

7 - Para criar um menu, é necessário declarar obrigatoriamente objetos das seguintes classes

Selecione uma ou mais opções:

- **MenuBar**
- **Menu**
- **MenuItem**

Resumindo (Sumários dos slides)

Classes abstratas e Interfaces

Classes abstratas

- Definem uma entidade abstrata;
- Representam entidades abstratas das quais nunca serão instanciados objetos;
- Podem ter métodos abstratos e métodos concretos;
- Os métodos abstratos definem um comportamento que deve ser implementado na hierarquia de classes.

Interfaces

- Definem um comportamento;
- Representam conjuntos de funcionalidades sem implementação;
- Apenas têm métodos abstratos;
- Os métodos das interfaces definem um comportamento que pode ser implementado por quaisquer classes (pertencendo ou não a uma hierarquia).

Exceções

Exceções mais comuns:

- **ArithmeticException** - Indica falhas no processamento aritmético, tal como uma divisão inteira por 0.
- **ArrayIndexOutOfBoundsException** - Indica a tentativa de acesso a um elemento de um array fora dos seus limites: ou o índice é negativo ou maior ou igual ao tamanho do array.
- **IndexOutOfBoundsException** - Indica a tentativa de usar um índice fora do limite de uma tabela.
- **ArrayStoreException** - Indica a tentativa de armazenamento de um objeto inválido numa tabela.
- **NegativeArraySizeException** - Indica a tentativa de criar uma tabela com dimensão negativa.
- **StringIndexOutOfBoundsException** - Indica a tentativa de usar um índice numa string fora dos seus limites
- **NumberFormatException** - Indica a tentativa de conversão de uma string para um formato numérico, mas que o seu conteúdo não representava um número para aquele formato.
- **NullPointerException** - Indica que a instrução tentou usar null onde era necessária uma referência a um objeto

- **IllegalArgumentException** - Quando o argumento do método tem um valor impossível
- **IOException** - Indica a ocorrência de qualquer tipo de falha em operações de entrada e saída.

A linguagem Java permite o tratamento de situações de exceção de uma forma normalizada através da utilização de 5 palavras chave correspondentes a cláusulas especiais, a saber:

- **throws**
- **throw**
- **try**
- **catch**
- **finally**

O mecanismo de exceções é a forma indicada em programação orientada por objetos para lidar com os erros.

Introdução ao JavaFX

Estrutura de um programa em JavaFX e suas classes base:

- **Application**
 - o Lança os componentes da interface gráfica com o utilizador do JavaFX numa “thread” protegida.
- **Stage (Palco)**
 - o “Espaço” onde se desenrola a peça de teatro.
 - o Equivale a uma janela da aplicação.
- **Scene**
 - o Define os vários cenários que queremos apresentar numa janela.
 - o Em cada momento definimos o cenário a apresentar com o método `Stage.setScene()`.
- **Node**
 - o A classe abstrata e base dos nós de um grafo de cena.
- **Group**
 - o É uma classe de coleção destinada a agrupar objetos das subclasses de Node.

- **Escrita de texto**
 - o O texto escreve-se com a classe `javafx.scene.text.Text`
 - o E define-se a sua apresentação com a classe `javafx.scene.font.Font`
- **Propriedades**
 - o Todos os nós, possuem “propriedades” que permitem alterar o seu aspeto e comportamento
- **Desenho de Figuras Geométricas**
 - o A classe `javafx.scene.shape.Shape` e suas classes derivadas

JavaFX – Eventos e Painéis

- **Eventos**
 - A Programação baseada em eventos permite interagir com uma interface gráfica
 - A Classe **Event**
 - Uma ação do utilizador sobre um componente do GUI
 - Faz com que esse objeto gere um evento
 - Evento esse que, ao ser apanhado pelo “handler” apropriado, despoleta a execução do código desse “handler”
 - Para que um qualquer componente do GUI (nó) reaja a diferentes ações do utilizador basta criar um “handler” para cada uma dessas ações no respetivo componente.
- **Painéis (Pane)**
 - `BorderPane` (cinco zonas: top, bottom, left, right, center)
 - `GridPane` (uma grelha)
 - `HBox` e `VBox` (dispõem os componentes horizontal e verticalmente)
 - Painéis compostos por painéis (podemos sempre inserir painéis em painéis)

JavaFX – Controlos

Controlos

- **Button** – inserir imagem num botão
 - 1 - Criar uma imagem associada a um ficheiro (Image).
 - 2 - Associar a imagem ao botão (ImageView).
- **TextField** – criação e utilização
 - 1 - Criar um objeto do tipo TextField.
 - 2 - Usar getText() para ler conteúdo e o setText() para o alterar.
- **Label** – criação e utilização
 - 1 - Criar um objeto do tipo Label.
- **ListView** – criação, utilização e preenchimento
 - Criar um objeto do tipo ListView.
 - Criar uma ObservableList e associá-la à ListView (será utilizado o toString)
 - Adicionar ou remover linhas à ObservableList para preencher a ListView
- **Uso de Canvas e GraphicsContext**

JavaFX – Propriedades

- Criação de interfaces gráficas através do acrescento de novas classes que permitem a visualização dos objetos já existentes no domínio do problema.
- Para cada classe poderá ser criado um "visualizador" (implementa as operações básicas – CRUD: Create, Read, Update e Delete).
- A criação dos visualizadores é feita por herança de objetos gráficos já existentes. Uma vez que se pretende apresentar informação dos vários atributos, recorre-se, normalmente, à extensão das classes que permitem agrupar/apresentar diversos elementos gráficos – os painéis. No respetivo construtor (que recebe o objeto a apresentar) são criados todos os elementos gráficos necessários.
- A funcionalidade fica completa através da criação dos diversos EventHandler e da ligação (binding) entre as propriedades do objeto e a dos elementos gráficos. Esta ligação pode ser feita de forma bidirecional (bindBidirectional) ou unidirecional (bind), para sincronizar os valores ou genericamente através da criação de um Listener.

- **ListView e ComboBox**
 - Permitem apresentar e manipular coleções de elementos.
 - A ListView permite seleção Múltipla ou Simples.
 - A ComboBox só permite a seleção de um elemento (Simples).
- **Tab**
 - Através de um controlo TabPane, podemos implementar a alternância entre vários painéis.
 - Associando um painel a um Tab e por sua vez os vários Tabs a um TabPane.
- **Accordion**
 - Através de um controlo Accordion, também é possível implementar a alternância entre painéis.
 - Criando os vários TitledPane e associando-os ao Accordion.
- **Menu**
 - É possível definir uma hierarquia de menus e submenus, usando as classes MenuBar, Menu, MenuItem, CheckMenuItem, RadioMenuItem, SeparatorMenuItem.
- **CheckBox**
 - Através de um controlo CheckBox, podemos implementar a seleção de uma opção.
- **RadioButton**
 - Para implementar a escolha exclusiva através de RadioButton, temos de definir um ToggleGroup e associar cada RadioButton ao grupo criado.

JavaFX – Janelas e Formas

- **Criação de novas janelas**
 - **Diálogos**
 - Criação de Diálogos
 - FileChooser e DirectoryChooser
- **Formas**
 - O package `javafx.scene.shape.*`;
 - A classe abstrata `javafx.scene.shape.Shape`;
- **Cores**
 - O package `javafx.scene.paint.*`;
 - A classe `javafx.scene.paint.Color`;
 - A aplicação de cores a uma Shape passa por duas etapas:
 - 1 - Criar a cor ou o gradiente com o construtor da classe;
 - 2 - Alterar a propriedade a colorir com o `setXxx` da Shape.
- Databinding para modificar os valores das propriedades dos objetos