

Exame Modelo de PPROG

Licenciatura em Engenharia Informática

Instituto Superior de Engenharia do Porto

Junho de 2021 - Exame sem consulta - Duração: 1 hora (sem tolerância)

1. [2 valores]

Considerando o seguinte excerto de código:

```
1  enum Level {
2      LOWEST(0), LOW(10), MED(20), HIGH(50), HIGHEST(100);
3      private int value;
4
5      private Level(int value) {
6          this.value = value;
7      }
8
9      public int getValue() {
10         return value;
11     }
12 }
13
14 public class Enum_Q1 {
15
16     public static void main(String[] args) {
17         Level lev1 = Level.LOWEST;
18         Level lev2 = Level.LOW;
19         Level lev3 = Level.MED;
20         Level lev4 = Level.HIGH;
21         Level lev5 = Level.HIGHEST;
22
23         int res = A.getValue() * B.ordinal();
24         System.out.println(res);
25     }
26 }
```

Indique quais deveriam ser os objetos A e B para que a saída resultante da sua execução seja igual a 80.

A=

B=

2. [4/3 valores]

Das afirmações que se seguem, selecione as verdadeiras:

Selecione uma ou mais opções de resposta:

- O conceito de herança estabelece que uma subclasse possa aproveitar os atributos e métodos de uma superclasse. A subclasse pode ter acesso aos métodos e atributos públicos e protegidos da superclasse. O polimorfismo é aplicado ao caso em que existe a necessidade de implementar métodos sobrecarregados, nos quais a subclasse necessita de implementar dois métodos com o mesmo nome e parâmetros diferentes.
- Atributos e métodos podem ser reaproveitados usando herança, quando uma subclasse herda as características de uma superclasse. Uma subclasse pode ter acesso aos membros de uma superclasse, independente do modificador de acesso usado. O polimorfismo é um recurso que permite a uma subclasse reescrever os métodos herdados de uma superclasse, sendo este método abstrato ou não.
- A herança permite que os membros de uma superclasse, possam ser reaproveitados na definição de uma subclasse. Esta subclasse tem acesso aos membros públicos e protegidos da superclasse. O polimorfismo, associado à herança, permite que métodos abstratos definidos numa classe abstrata sejam implementados nas subclasses, podendo estes métodos, nas subclasses, apresentar comportamentos distintos.
- A herança e o polimorfismo são complementares, ou seja, devem ser aplicados em conjunto. A herança existe a partir de classes abstratas que contêm atributos e métodos abstratos. O polimorfismo obriga que subclasses implementem os métodos e atributos da superclasse. O acesso aos atributos da superclasse é independente do modificador de acesso usado.
- Nenhuma das outras opções.

3. [4/3 valores]

Nos conceitos presentes no paradigma orientado a objetos, **classe(s)** é uma estrutura composta por **atributo(s)** que descrevem as suas propriedades e também por **método(s)** que definem o seu comportamento. **objeto(s)** são **instância(s)** dessa estrutura e só existem em tempo de execução.

4. [2/3 valores]

Mediante a execução do seguinte programa:

```
①  class A {  
12  
13      int i;  
14  
②  [-]  void display() {  
16      System.out.println(i);  
17  }  
18 }  
19  
20 class B extends A {  
21  
22     int j;  
23  
③  [-]  void display() {  
25      System.out.println(j);  
26  }  
27 }  
28  
29 public class NewMain {  
30  
31 [-]  public static void main(String[] args) {  
32     B obj = new B();  
33     obj.j = 2;  
34     obj.display();  
35  }  
36 }
```

Selecione uma ou mais opções de resposta:

- Erro de compilação (Compilation error)
- Nenhuma das outras opções
- 1
- 0

5. [2/3 valores]

Entre os exemplos de código que se seguem selecione os que compilam sem erros.

Selecione uma ou mais opções de resposta:

```
1 class A3 {  
2     public static void aMethod() {  
3         System.out.println("Static method in A3");  
4     }  
5 }  
6  
7 class B3 extends A3 {  
8     public static void bMethod() {  
9         System.out.println("Calling static method of superclass");  
10        super.aMethod();  
11    }  
12 }
```

```
1 class A5 {  
2     private static int intVar = 10;  
3     public static void metodo1() {  
4         System.out.println("intVar: " + intVar);  
5     }  
6     public int metodo2() {  
7         metodo1();  
8         intVar += 1;  
9         return intVar;  
10    }  
11 }
```

```
1 class A6 {  
2     private int intVar = 10;  
3     public static void metodo() {  
4         System.out.println("intVar: " + intVar);  
5     }  
6 }
```

```
1 class A4 {  
2     private static int intVar = 10;  
3     public static void metodo1() {  
4         System.out.println("intVar: " + intVar);  
5     }  
6     private static int metodo2() {  
7         metodo1();  
8         intVar += 1;  
9         return intVar;  
10    }  
11 }
```



falta o parâmetro



6. [2/3 valores]

Qual/quais das seguintes situações não corresponde a "overloading" (sobrecarga de métodos)?

Selecione uma ou mais opções de resposta:

- Mais do que um método com o mesmo nome, o mesmo número de parâmetros mas tipos de parâmetros diferentes.
- Mais do que um método com o mesmo nome mas com número diferente de parâmetros.
- Mais do que um método com o mesmo nome, o mesmo número de parâmetros e os mesmos tipos de parâmetros.
- Mais do que um método com o mesmo nome mas com número diferente de parâmetros ou com tipos de parâmetros diferentes.

7. [2 valores]

Considere o seguinte extrato de código da classe `Picture`. `Shape` é uma interface.

```
1 public class Picture {  
2     List<Shape> drawList = new ArrayList<>();  
3     void init() {  
4         Rectangle rect1=new Rectangle (new Point(10,10),4,8);  
5         Circumference c1=new Circumference (new Point (0,0), 6);  
6         drawList.add(c1);  
7         drawList.add(rect1);  
8     }  
9     void drawing(int x, int y) {  
10        for (int i=0;i< drawList.size()-1;i++)  
11            if (drawList.get(i).greaterThan(drawList.get(i+1)))  
12                drawList.get(i).toDraw(); 1*  
13            else drawList.get(i).moveTo(x,y); 2*  
14    }  
15 }
```

Diga quais os métodos que a interface `Shape` publica.

Selecione uma ou mais opções de resposta:

- int size()
- void moveTo()
- Nenhuma das outras.
- void toDraw() 1*
- boolean greaterThan(Shape shape) 2*

Pois têm parâmetros

8. [4/3 valores]

Considere o extrato de código seguinte.

```
class Animal { /* ... */ }
class Dog extends Animal { /* ... */ }

void print(List<? extends Animal> list) {
    //missing_code
    System.out.println(e);
}
```

Qual(is) da(s) instrução(ões) seguinte(s) pode(m) substituir o comentário *//missing_code*?

Selecione uma ou mais opções de resposta:

- for (Dog e : list)
- for (? e : list)
- for (<? extends Animal> e : list)
- for (Animal e : list)
- for (Object e : list)

9. [4/3 valores]

Complete o código:

```
public class A implements Comparable {

    ...

    public int compareTo ( Object obj ) {
        A o = ( A ) obj;
        if ( m1() > o.m1() ) return 1 ;
        else if ( m1() < o.m1() ) return -1 ;
        else return 0
    }
}
```

10. [2/3 valores]

Sendo A um objeto que possui uma relação do tipo ``has-a'' com um objeto B (A has-a B), B deixa de existir se A é eliminado. Neste caso, estamos perante uma relação de:

Selecione uma ou mais opções de resposta:

- Associação.
- Composição.
- Encapsulamento.
- Agregação.

11. [2/3 valores]

Para criar uma lista de objetos da classe Employee, qual o código que utiliza parâmetros de tipo de forma correta?

Selecione uma ou mais opções de resposta:

- List <> lstEmployee = new ArrayList <> () X
- List <Employee> lstEmployee = new ArrayList (Employee) X
- List lstEmployee = new ArrayList <Employee> ()
- List <Employee> lstEmployee = new ArrayList <> ()
- List <Employee> lstEmployee = new ArrayList() X

12. [2/3 valores]

A interface `SomethingWrong` possui métodos mal declarados. Identifique-os.

```
1 | public interface SomethingWrong {  
2 |     void v();  
3 |     abstract String s();  
4 |     static int i();  
5 |     public float f();  
6 |     final double d();  
7 | }
```

Selecione uma ou mais opções de resposta:

- d()
- i()
- v()
- f()
- s()

13. [2/3 valores]

Qual é a saída do seguinte programa?

```
1 | public class Foo  
2 | {  
3 |     public static void main(String[] args)  
4 |     {  
5 |         try  
6 |         {  
7 |             return;  
8 |         }  
9 |         finally  
10 |         {  
11 |             System.out.println( "Finally" );  
12 |         }  
13 |     }  
14 | }  
15 | }
```

Selecione uma ou mais opções de resposta:

- O código é executado sem dar origem a qualquer saída.
- Durante a execução é lançada uma exceção.
- Finally
- A compilação falha.

Pois, o finally

é executado

sempre após o try!

14. [2/3 valores]

Qual/quais destes métodos retorna a descrição de uma exceção?

Selecione uma ou mais opções de resposta:

- obtainDescription()
- obtainException()
- getException()
- getMessage()

15. [4/3 valores]

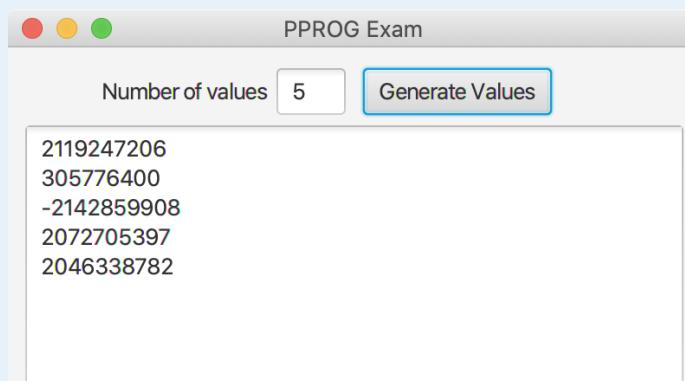
No JavaFX, um objeto do tipo Scene:

Selecione uma ou mais opções de resposta:

- Tem que ter sempre um componente Button ou Menu para que possa ser executada alguma ação.
- Só pode ser usado num único Stage.
- Não pode ter eventos associados.
- Pode estar associado a vários controllers existentes na camada controller da arquitetura MVC.
- Pode apresentar vários tipos de componentes/controlos.
- Tem que ter associado um fx:id para cada componente/controlo existente.
- Pode ser usado para disponibilizar mecanismos de interação com o utilizador.

16. [2 valores]

Pretende-se uma aplicação que apresente numa interface gráfica uma quantidade de valores aleatórios, definida pelo utilizador.



Assuma que:

1. A classe **RandomValues** possui o seguinte método **public static final String generateRandomValuesList(int nr_random_value)**, que cria e retorna uma string com **nr_random_value** valores aleatórios (um por cada linha). Esta é uma classe de modelo.
2. Na classe da interface gráfica, o controlo (**TextField**) onde é especificado o "Number of values" (na imagem tem o valor "5") tem um fx:id **txtNumberValues** e o controlo onde os valores são apresentados (**TextArea**) tem um fx:id **txtAreaRandomValues**.

Implemente o método com a assinatura **private void generateListAction(ActionEvent event)**, que é invocado quando for clicado o botão "Generate Values". Assuma que o valor do **TextField** é sempre um valor inteiro positivo. Tenha em consideração a **arquitetura MVC** na implementação da solução.

```
private void generateListAction(ActionEvent event) {  
}  
-----  
//implemente aqui outras funcionalidades (classes e/ou métodos) que entenda necessárias
```

17. [2/3 valores]

O que é um atributo *transient*?

Selecione uma ou mais opções de resposta:

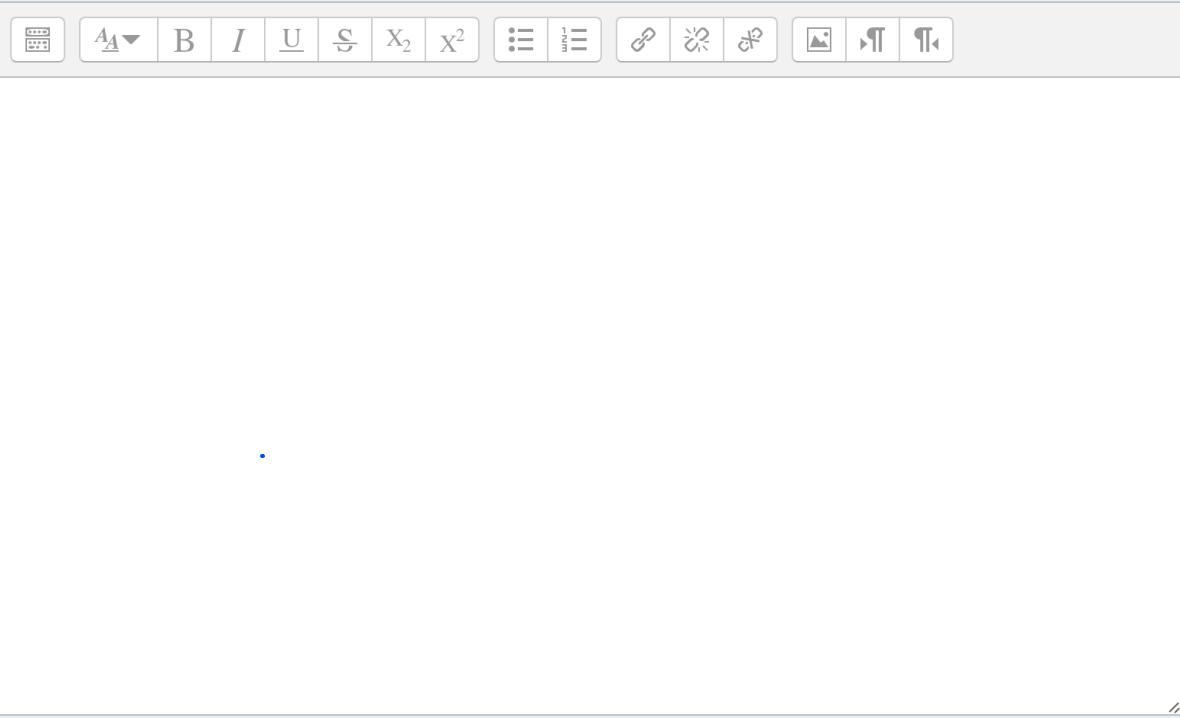
- É um modificador utilizado para declarar classes temporárias.
- Um atributo *transient* é um atributo que é serializado durante o processo de serialização (*Serialization*).
- Um atributo *transient* é um atributo que não é serializado durante o processo de serialização (*Serialization*).
- Um atributo *transient* é um atributo que é marcado como serializável (*Serializable*).

18. [4/3 valores]

Considere a declaração das seguintes classes:

```
1 public class Address {  
2     private String street;  
3     private String city;  
4     private int doorNumber;  
5     private String zipCode;  
6 }  
7  
8 public class Person implements Serializable{  
9     private String name;  
10    private int age;  
11    private Address adress;  
12 }
```

Considere que a classe `Address` **não pode implementar a interface Serializable**. Defina o código necessário para que quando da serialização de um objeto da classe `Person` seja também serializado o objeto correspondente da classe `Address`. Identifique a classe em que esse código deverá ser introduzido. Considere já implementados os métodos *getter* das classes.



The form consists of a large central text area for writing the answer, with a toolbar at the top featuring various icons for text styling and document operations. The toolbar includes icons for bold, italic, underline, superscript, subscript, font size, alignment, and other document functions.