

Programação Orientada por Objetos

1º Teste, 4 de Maio 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: (4 Valores)

1.1 (0.5) A herança orientada por objetos modela um relacionamento do tipo:

- a) "tem um".
- b) "é uma espécie de".
- c) "quer ser".
- d) "contém".

1.2 (0.5) Para as classes A e B definidas a seguir:

```
class Thing {  
    public int f() {return (5);}  
    public static int g() {return (6);}  
}
```

```
class Stuff extends Thing {  
    public int f() {return (2);}  
    public static int g() {return (4);}  
}
```

O que irá mostrar o código seguinte?

```
Stuff s = new Stuff ();  
Thing t = s;  
System.out.println(t.f() * t.g());
```

- a) 30
- b) 20
- c) 8
- d) 12

1.3 (0.5) Qual das seguintes opções é falsa?

- a) Uma classe que contém métodos abstratos é uma classe abstrata.
- b) Os métodos abstratos só podem ser implementados numa classe derivada.
- c) Uma classe abstrata não pode ter métodos que não sejam abstratos.
- d) Uma classe deve ser qualificada como classe "abstrata", se contiver um método abstrato.

1.4 (0.5) Na orientação por Objetos, qual é a afirmação que melhor se ajusta à definição de uma interface?

- a) É um método vazio de uma classe.
- b) É um meio dado ao utilizador de uma aplicação Orientada por Objetos de interagir com ela.
- c) É o conjunto de assinaturas de operações públicas de uma classe.
- d) É o conjunto de atributos públicos de uma classe.

1.5 (0.5) Escolha a afirmação **correta**:

- a) O tipo estático de uma variável é o tipo do objeto referido por ela, enquanto o tipo dinâmico é o tipo declarado.
- b) No polimorfismo e pelo princípio da substituição, o tipo declarado de uma variável pode ser o tipo da superclasse, enquanto o tipo dinâmico pode ser qualquer classe derivada da superclasse.
- c) O compilador verifica o tipo estático e o tipo dinâmico da invocação de um método.
- d) Quando na execução de um programa invocamos um método de um objeto, primeiramente é verificado se o método existe na superclasse.

- 1.6 (0.5) Para as classes e interfaces, os tipos de modificador de acesso disponíveis em Java são:
- a) *private* e *protected*.
 - b) *package private* (acesso por defeito) e *public*.
 - c) *private* e *public*.
 - d) *package private* (acesso por defeito) e *protected*.
- 1.7 (0.5) Quando um método redefinido é chamado dentro de outro método da mesma subclasse, o método que irá ser realmente chamado é o método da:
- a) superclasse
 - b) subclasse
 - c) o compilador escolherá aleatoriamente
 - d) o interpretador escolherá aleatoriamente
- 1.8 (0.5) Qual das seguintes afirmações é falsa:
- a) Uma classe concreta que implementa uma interface deve implementar todos os métodos da interface.
 - b) Uma interface pode implementar outra interface.
 - c) Uma interface pode estender outra interface.
 - d) Uma interface pode ser uma solução para a herança múltipla em java

Grupo 2: (7 Valores)

Pretende-se desenvolver um programa para uma clinica médica. Para esse efeito criaram-se os protótipos das classes **Person**, **Patient**, **Clinic**, **MedicalAct** e da interface **Employee** representados na **Figura 1**. O programa principal de teste (método **main** da classe **Grupos2e3**) e o respetivo *ouput* são apresentados na **Figura 2**.

- 2.1 (2.0) – Tendo em conta a classe **Patient** da **Figura 1**, complete-a fornecendo o código em falta, nomeadamente:
- a. O construtor sem argumentos.
 - b. O construtor que leva como argumentos o nome do paciente e a data de nascimento.
 - c. O método **setName** sabendo que no caso deste método receber uma *String* nula ou com menos de 3 caracteres deve ser fornecido como nome o texto “N/D” (Não definido).
 - d. O método **toString** tendo em conta o que é escrito no ecrã por este método de acordo com o que é mostrado no output do método **main** na **Figura 2**.
- 2.2 (2.5) – Escreva agora o código da classe **Doctor** que representa um médico, sabendo que:
- A classe **Doctor** implementa a interface **Employee**.
 - Possui dois atributos, um para o valor do salário mensal (**salary**) e outro para o número de membro da ordem dos médicos (**number**), tem também métodos seletores para estes atributos e o método modificador do atributo **salary** que não permite guardar valores de salário inferiores a zero.
 - Possui um único construtor que recebe o nome do médico e o número da ordem e que está a ser usado no método **main** na **Figura 2**.
 - Tem um método **toString** que retorna um texto com o nome do médico e o seu número e que deve estar de acordo com o que é mostrado no *output* do método **main** também na **Figura 2**.
- 2.3 (2.5) – Considere a classe **Clinic** que representa uma clinica médica e que guarda a informação de todos os seus trabalhadores e dos seus clientes de acordo com o que está representado na **Figura 1**. Defina para esta classe:
- a. O código do construtor.
 - b. O método **addWorker** que adiciona um empregado da clinica. Veja a sua utilização no método **main**.
 - c. O método **addClient** que adiciona um paciente. Veja a sua utilização no método **main**.
 - d. O método **usersList** que devolve um texto com a lista de todos os utentes da clinica (pacientes e empregados). Pode utilizar o método **users** disponibilizado na classe (o código deste método irá ser pedido noutra alínea, não é necessário agora). Veja a utilização do método **usersList** dentro do método **main** e o respetivo output produzido no ecrã na **Figura 2**.

Grupo 3: (6 Valores)

Continuando a desenvolver o programa anterior, considere agora a classe **MedicalAct** da **Figura 1**. Esta classe representa um ato médico que pode ser uma consulta (**Appointment**) ou uma pequena cirurgia (**Surgery**).

3.1 (2.0) – Tendo em conta o código da **Figura 2**, respeitante a este grupo, crie a classe **Appointment** fornecendo apenas o código necessário para que o programa compile sem erros. Defina nesta classe, usando uma constante, o custo de uma consulta como sendo de 50 Euros.

3.2 (1.0) – Considere agora o método **main** da **Figura 2**. Neste método são definidos 3 atos médicos. Escreva o código que deverá ser colocado no local assinalado com a referência a esta alínea. Tenha em conta o que é mostrado no output em frente ao comentário **// Alínea 3.2**. No código pedido, deve criar uma coleção onde irá adicionar os 3 atos médicos definidos e através do polimorfismo deve escrever o tipo de ato e o valor associado tal como é mostrado no *output*. No fim é necessário escrever o custo total de todos os atos médicos.

3.3 (1.5) – Crie o código do método **users** da classe **Clinic**. Este método retorna uma lista com todos os trabalhadores e clientes da clinica que estão guardados nas coleções **clients** e **workers**.

3.4 (1.5) – A classe **Clinic** guarda no atributo **appointments** um registo de todas as consultas de um cliente. Sendo assim, defina o método **addAppointment** dessa classe de forma a que seja adicionado ao paciente referido a consulta recebida no parâmetro. Atenção que esta deverá ser adicionada à lista de consultas para esse paciente. Se o paciente não existir, deve ser criada uma entrada para o paciente. Exemplifique a utilização do método adicionando o **act1** definido no método **main** à clinica **est** definida no mesmo método.

Grupo 4: (3 Valores)

4.1 (2.0) – Tendo em conta o protótipo da aplicação para a clinica, explique como faria para adicionar registos médicos para os pacientes (classe **ClinicalRecord**) onde constasse a informação dos resultados das várias medições efetuadas (peso, altura e tensão). Indique apenas classes e atributos que adicionaria e onde colocava os objetos adicionais, no caso de afetarem as classes existentes na aplicação. Indique também qualquer relação de herança que exista).

4.2 (1.0) – Escreva um protótipo da carta CRC da classe **ClinicalRecord** que irá ser adicionada à aplicação de acordo com a alínea anterior.

<pre> public abstract class Person { private String name; public Person(String name) { this.name = name; } public String getName() { return name; } public void setName(String name) { this.name = name; } public String toString() { return "Name: " + name; } } </pre>	<pre> // class Doctor - alínea 2.2 // // Implementa Employee // Tem os atributos number e salary // Tem um construtor que recebe o nome e o number // Tem métodos seletores para os atributos // Tem o método modificador do atributo salary // Tem a redefinição do método toString // public class Clinic { private String name; private HashSet<Patient> clients; private ArrayList<Employee> workers; private HashMap<Patient,ArrayList<Appointment>> appointments; public Clinic(String name) { // Alínea 2.3 a. } // Método addWorker - alínea 2.3 b. // Método addClient - alínea 2.3 c. public String usersList() { // Alínea 2.3 d. } private ArrayList<Person> users() { // Alínea 3.3 } public void addAppointment(Appointment appointment) { // Alínea 3.4 } } </pre>
<pre> public class Patient extends Person { private LocalDate birthDate; public Patient() { // Alínea 2.1 a. } public Patient(String name, LocalDate birthDate) { // Alínea 2.1 b. } public LocalDate getBirthDate() { return birthDate; } public void setBirthDate(LocalDate birthDate) { this.birthDate = birthDate; } public int getAge() { return Period.between(birthDate, LocalDate.now()) .getYears(); } public void setName(String name) { // Alínea 2.1 c. } public String toString() { // Alínea 2.1 d. } public int hashCode(){ //Código omitido } public boolean equals(Object obj) { // Código omitido } } </pre>	<pre> public abstract class MedicalAct { private Doctor doctor; private Patient patient; public MedicalAct(Doctor doctor, Patient patient){ this.doctor = doctor; this.patient = patient; } public Doctor getDoctor() { return doctor; } public Patient getPatient() { return patient; } public abstract double getPrice(); public abstract String getDescricao(); } </pre>
<pre> public interface Employee { double getSalary(); String getName(); } </pre>	

Figura 1: Classes **Person**, **Patient**, **Clinic**, **MedicalAct** e interface **Employee**

<pre> public class Grupos2e3 { public static void main(String[] args) { // Grupo 2 Patient john = new Patient("John", LocalDate.of(2000, 5, 4)); System.out.println(john); Doctor drAnne = new Doctor("Anne", 1234); drAnne.SetSalary(5000.0); System.out.println(drAnne); Clinic est = new Clinic("ESTClinic"); est.addWorker(drAnne); est.addClient(john); System.out.println("\nUtentes:"); System.out.println(est.usersList()); // Grupo 3 MedicalAct act1 = new Appointment(drAnne, john, LocalDateTime.of(2019, 5, 6, 10, 20)); MedicalAct act2 = new Surgery(drAnne, john, LocalDate.of(2019, 5, 7)); MedicalAct act3 = new Appointment(drAnne, john, LocalDateTime.of(2019, 5, 8, 11, 20)); // Alínea 3.2 // Alínea 3.4 - adicionar Appointment (Consulta act1) } </pre>	<p><i>Output do método main:</i></p> <p>Name: John Data Nascimento: 2000-05-04</p> <p>Dr. Anne (nr.1234)</p> <p>Utentes: John Anne</p> <p>Consulta: 50.0 Operação: 300.0 Consulta: 50.0 Custo Total: 400.0</p>
---	--

Figura 2: método main e classe Grupos2e3