Prova de Programação II
Professor Bernardo Henz

Nome:

1) Qual dos seguintes itens **não** é uma vantagem do Paradigma de Orientação a Objetos
a) Modularização
b) Clareza do código
c) Velocidade de execução
d) Reutilização de código

2) A respeito de conceitos do POO, quando falamos que "é o princípio pelo qual métodos com a mesma assinatura podem se comportar de maneira diferente". Esta é a definição de:
a) Herança
b) Abstração
c) Polimorfismo
d) Agrupamento

3) A respeito de classes e objetos na linguagem Java, complete as lacunas:
" A palavra _____ é utilizada para definirmos uma classe, e a palavra ____ é utilizada para instanciarmos um objeto de determinada classe"
a) new, class
b) new class, Instance
c) class, new
d) new class, new

4) Considere o código a seguir, e assinale a alternativa **falsa**:

```
public static void main(String[] args) {
    Carro c1 = new Carro( );
    c1.marca = "Volkswagen";
    c1.andar( );
    }
```

a) 'marca' é um atributo (provavelmente do tipo String) de 'c1'
b) 'c1' é uma classe com obrigatoriamente 1 atributo e 1 método
c) 'andar' é um método que não recebe nenhum parâmetro
d) Se o código roda sem erro, então 'marca' e 'andar' possuem o modificador de acesso 'public'

5) Considerando o código a seguir, assinale a alternativa **verdadeira**:

```
public static void main(String[] args) {
  Carro c1 = new Carro( );
  Carro c2 = new Carro("Audi", 1992);
  Carro c3 = c1;
  c3.setMarca("Tesla");
}
```

a) Neste exemplo são instanciados 3 objetos distintos, cada um criado de uma maneira distinta
b) A marca do objeto referenciado por 'c1' depende de como este atributo foi inicializado no construtor vazio ('*Carro ( )*')
c) Métodos 'set' (como em 'setMarca') são bastante utilizados para modificar atributos com modificadores de acesso que não são 'public'
d) Para modificar a marca do Carro c2 para Fiat, basta utilizarmos o código '*c2.marca = "Fiat"* '

6) A respeito de construtores, assinale a alternativa **correta**:
a) Construtor é sempre um método sem retorno e sem parâmetros
b) Um construtor sempre tem o mesmo nome da classe, e não é possível uma mesma classe ter mais de 2 construtores
c) A palavra 'super' é utilizada para chamarmos o construtor vazio da classe pai
d) Para reutilização de código, podemos chamar um construtor (da mesma classe ou da classe pai) dentro de outro construtor

7) Considere o seguinte código em Java:

```
public class Carro {
  private String marca;
  private int ano;
  Carro(String marca, int ano) {
       ____.marca = marca;
       ____.ano = ano;
   }
}
```

Assinale qual alternativa completa corretamente as lacunas:
a) this
b) self
c) super
d) Carro

8) Considere a classe Carro do exercício anterior, e complete as lacunas do seguinte método:

```
 [1] getMaisAntigo(Carro c1, Carro c2) {
     if ( c1.  [2]  ( )  [3]  c2.  [2]  ( ) )
          return  [4] ;
     else
          return c2;
}
```

Qual a alternativa que corretamente completa, respectivamente, as lacunas [1], [2], [3] e [4] ?
a) public Carro, getAno, < , c1
b) public void, getAno, >=, c1.getAno()
c) public Carro, ano, <, c1
d) static public Carro, getAno, <= , c1

9) Considere o código abaixo:

```
public class Pessoa{
     private String nome;
     private Carro meuCarro;
     public void registrarCarro(Carro c1){
          meuCarro = c1;
     }
     public Carro getCarro( ){
          return meuCarro;
     }
}
```

Assinale a alternativa **errada**:
a) O método 'getCarro( )' não trata ponteiro null e dará erro quando for chamado neste caso.
b) Não é possível registrar mais de 1 Carro por pessoa

c) Como o atributo ‘meuCarro’ é private, temos que ter métodos get e set para modificar esse atributo (neste exemplo, o método ‘registrarCarro’ funciona como método ‘set’)
d) Para que uma Pessoa possa ter mais de um Carro, podemos utilizar alguma Collection de Java, como ArrayList ou LinkedList

10) Gabriela não estava feliz com a implementação acima, pois não permitia que 1 Pessoa tivesse mais de 1 Carro. Então, ela resolveu utilizar uma ArrayList<Carro> para armazenar a lista de carros de cada pessoa. Ela conseguiu implementar corretamente o construtor da classe (omitido no código abaixo), mas está com dificuldades para readaptar os demais métodos de acordo. Ajude-a completando as lacunas:

```
public class Pessoa{
    private ArrayList<Carro> meusCarros;
    public void registrarCarro(Carro c1){
        meusCarros. [1] (c1);
    }
    public Carro getCarro(int index){
        if (index [2] meusCarros. [3] ())
            return meusCarros. [4] (index);
        else
            return null;
    }
}
```

A alternativa que corretamente completa, respectivamente, as lacunas [1], [2], [3] e [4] é:
a) add, > , length, get
b) put, <= , size, remove
c) add, <, size, get
d) put, >=, length, get

11) Gabriela quer ir além e conseguir retornar o valor total de todos os carros de uma pessoa. Para isso ela modificou o código de Carro para ter o atributo valor, e o métodos getValor:

```
public class Carro {
  private float valor;
  public float getValor(){
        return valor;
  }
}
```

Agora você deve ajudar ela a implementar o método getValorTotalDosCarros( ). Este método é implementado dentro da classe Pessoa. Qual das seguintes implementações implementa corretamente o método?
a)

```
public float getValorTotalDosCarros(){
   float somatorio = 0;
   for(int i=0; i<= meusCarros.length(); i++)
```

```
            somatorio += meusCarros.get(i).getValor();
        return somatorio;
       }
```
b)
```
public float getValorTotalDosCarros(){
        float somatorio = meusCarros.get(0).getValor();
        for(int i=1; i<= meusCarros.length(); i++)
            somatorio += meusCarros.get(i).getValor();
        return somatorio;
       }
```
c)
```
public float getValorTotalDosCarros(){
        float somatorio = 0;
        while ( !meusCarros.empty() )
            somatorio += meusCarros.remove(i).getValor();
        return somatorio;
       }
```
d)
```
public float getValorTotalDosCarros(){
        float somatorio = 0;
        for(int i=0; i<= meusCarros.size(); i++)
            somatorio += meusCarros.get(i).getValor();
        return somatorio;
       }
```

12) A respeito de herança na linguiagem Java, assinale a alternativa **correta**:

a) A subclasse herda características e comportamentos da superclasse, podendo acessar todos os atributos da superclasse

b) O termo abstract quando utilizado na frente do método da subclasse, obriga a superclasse a implementar este método

c) A palavra extends é utilizada para indicar que uma subclasse herda uma superclasse, mesmo que essa superclasse seja uma classe abstrata

d) Uma subclasse pode herdar várias superclasses, basta separarmos cada superclasse após o extends com vírgula

Utilize a seguinte implementação para responder as perguntas 13, 14 e 15:

```
public abstract class ServidorPublico{
    protected String codigo;
    private float salario;
    public abstract void trabalhar( );
    public float getSalario( ){
        return salario;
    }
}

public class Concursado extends ServidorPublico {
    protected Edital concursoIngresso;
    public void trabalhar( ){
        ArrayList<Funcoes> listaFuncoes = concursoIngresso.getAtribuicoes( );
        for (int i=0; i< listaFuncoes.size(); i++)
            listaFuncoes.get(i).desempenhar();
```

```
        }
    }

    public class Comissionado extends ServidorPublico{
        protected float bonificacao;
        public float getSalario( ){
            return salario+bonificacao;
        }
    }
```

13) Com respeito a implementação de 'ServidorPublico', assinale a alternativa **correta**:
a) Podemos criar um 'ArrayList<Concursado>', onde podemos adicionar diversos objetos do tipo 'ServidorPublico'
b) O código não funcionará. Como 'ServidorPublico' é uma classe abstrata, todos os seus métodos devem ser abstratos.
c) Como o atributo 'salario' é 'private', classes filhas não poderão acessar diretamente este atributo
d) O código está com erro de implementação, pois tem um método vazio 'trabalhar'

14) Com relação a implementação de 'Concursado', assinale a alternativa **correta**:
a) A implementação está errada. O 'extends' deve ser substituído por 'implements', visto que ' ServidorPublico' é uma classe abstrata
b) Fora o construtor estar omitido do código mostrado, a implementação está correta
c) Há erro de implementação, pois o método 'getSalario' não foi reimplementado
d) A implementação está errada: o método 'trabalhar' deveria retornar o número de horas trabalhadas

15) Com relação a implementação de 'Comissionado', assinale a alternativa **correta**:
a) Há erro de implementação, pois o método 'trabalhar' não é implementado, e também porque o atributo 'salario' é 'private' da superclasse
b) Fora o construtor estar omitido do código mostrado, a implementação está correta
c) Não é possível reescrever o método 'getSalario', pois ele não é abstrato
d) A implementação está errada. O 'extends' deve ser substituído por 'implements', visto que ' ServidorPublico' é uma classe abstrata

16) A respeito de Exceptions em Java, assinale a alternativa **correta**:
a) Exception é um bug que interrompe o funcionamento de um programa
b) stacktrace é uma lista de todas as instruções executadas ao rodar o código
c) Quando uma nova Exception é criada e lançada, ela deve ser tratada ou é enviada ao método logo abaixo da pilha de execução
d) Java é a única linguagem que existe Exception

17) A respeito de Exceptions em Java, complete as lacunas:
" _[1]_ e _[2]_ são utilizados para tentar rodar um trecho de código e tratar uma Exception caso ela aconteça. _[3]_ é utilizado para informar que determinado método pode lançar uma Exception de determinado tipo. _[4]_ é utilizado para criar e lançar uma Exception de determinado tipo"
A alternativa que completa, respectivamente, as lacunas [1], [2], [3] e [4] é:
a) try, throw, catch, throws new
b) catch, throw, try, throws new
c) throws, catch, try, throw new
d) try, catch, throws, throw new

18) A respeito de *Naming Conventions* em Java, assinale a alternativa correta:
a) Utilizamos nomes em caixa alta (exemplo: SALARIO) em atributos que são de classe (*static)*

b) A convenção camelCase utiliza underlines (_) para separar palavras
c) Nomes de classes iniciam com letra maiúscula e seguem a convenção UpperCamelCase
d) Métodos devem iniciar com letras maiúscula, e utilizar o snake_case

19) A respeito do uso de *static* e *final*, assinale a alternativa **correta**:
a) *final* é utilizado para indicar que determinado atributo/método pertence a classe
b) *static* é usado para indicar que determinado atributo/método pertence a classe. É necessário uma instância da classe para acessar tais atributos/métodos
c) *final*, quando utilizado antes de atributos, indica uma constante, que não pode modificar seu valor após atribuição
d) *static*, quando utilizado antes de classe, indica que a classe não pode ser herdada

20) A respeito de interfaces em Java, assinale a alternativa **incorreta**:
a) Interface é uma classe abstrata com métodos vazios, de maneira que não podem ser instanciadas
b) Utilizamos a palavra *implements* para indicar que uma classe implementa os métodos de uma Interface
c) Quando uma classe implementa uma Interface, a implementação dos métodos da Interface é opcional
d) Uma única classe é capaz de implementar mais de uma Interface