Aluno(a): _____

1. A partir das afirmativas abaixo relativas aos comandos da linguagem **Java**, pode-se afirmar que:

   I. A palavra chave *class* é usada para definir tipos derivados em **Java**. De fato, tais tipos constituem às principais estruturas dos programas orientados a objetos.

   II. O modificador *private*, quando aplicado a um atributo de classe, impede que qualquer outra classe tenha acesso a tal atributo.

   III. A palavra chave *extends* é usada para permitir a herança múltipla em **Java**.

   IV. O modificador *protected*, quando aplicado a um atributo de classe, permite que classes filhas desta classe, tenham acesso a este atributo.

   Estão corretas apenas as afirmativas:

   (a) II e IV (c) II e III (e) II, III e IV
   (b) I, II e IV (d) I e IV

2. Sobre as características da linguagem **Java**, analise as afirmativas a seguir.

   I. É uma linguagem fortemente orientada a objetos.

   II. Permite herança múltipla de classes, fornecendo maior flexibilidade e possibilidades de reaproveitamento de código.

   III. Requer a existência de uma máquina virtual para rodar.

   Está correto o que se afirma em:

   (a) I, apenas.
   (b) II, apenas.
   (c) III, apenas.
   (d) I e III, apenas. item I, II e III.

3. Em uma aplicação Java orientada a objetos que usa relações de herança, uma

   (a) superclasse não pode ter métodos sobrecarregados ou sobrescritos.

   (b) subclasse não pode ter mais que um construtor, mesmo que receba parâmetros diferentes.

   (c) subclasse normalmente usa a anotação @Override para indicar que um método da superclasse foi sobrescrito.

   (d) subclasse herda somente os atributos e os métodos privados da superclasse.

   (e) subclasse não pode sobrescrever um método da superclasse, mas o contrário é permitido.

4. O mecanismo de herança em **Java** é uma forma de reutilização de software na qual uma nova classe é criada, absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas. Sobre herança, analise as assertivas e, em seguida, assinale a alternativa que apresenta as corretas.

   I. No caso de herança simples, uma classe é derivada de uma superclasse indireta.

   II. A superclasse direta é a superclasse a partir da qual a subclasse herda explicitamente.

   III. A superclasse indireta é qualquer superclasse acima da classe direta na hierarquia de classe.

   IV. Os relacionamentos de herança formam estruturas hierárquicas do tipo árvore.

   (a) Apenas II, III e IV. (d) Apenas I e II.
   (b) Apenas III e IV.
   (c) Apenas I, II e IV. (e) Apenas I, II e III.

5. Considere as seguintes declarações, em um aplicativo **Java**:

```
class A {
     void metodo(int x) {
           System.out.out.println("x = " + (x-1));

     }

}

class B extends A {
     void metodo(int x) {
         super.metodo(x+1);
     }


}
```

   Quanto à sequência de comandos

```
A y = new B(); // Linha 1
y.metodo (5); // Linha 2
```

   pode-se afirmar que:

   (a) produzirá um erro de compilação na Linha 2.

   (b) ao ser executada, escreverá na saída padrão a mensagem x = 5.

   (c) produzirá um erro de compilação na Linha 1.

   (d) ao ser executada, escreverá na saída padrão a mensagem x = 6.

   (e) ao ser executada, escreverá na saída padrão a mensagem x = 4.

6. Em programação orientada a objetos, dizer que a **classe A** estende a **classe B** é o mesmo que dizer que:

   (a) a classe B é subclasse de A;

   (b) a classe A é superclasse de B;

   (c) a classe A é derivada de B;

   (d) a classe B é derivada de A;

   (e) as classes A e B são irmãs.

7. O estabelecimento de relações de herança na programação orientada a objeto permite o reúso de código. Na linguagem de programação **Java**,

   (a) para que uma **classe X** herde ciclicamente dela mesma, basta na declaração da **classe X** colocar a instrução ***extends* X**.

   (b) em uma relação de herança, uma classe herda os atributos de outra, mas não seus métodos.

   (c) em uma relação de herança, não pode haver sobrecarga de métodos nas subclasses.

   (d) um método de uma superclasse pode ser sobrescrito em suas subclasses.

   (e) para uma **classe X** herdar de uma **classe Y** e de uma **classe Z**, deve-se utilizar na declaração da **classe X**, a instrução ***extends* Y, Z**.

8. Analisando o código abaixo, é possível afirmar que:

```
public class A {
    public int c;
    private String d;
}

public class B extends A {
    private boolean e;

    public void g() {
        c = 5;
        d = "UEMG";
    }

}

public class C {
    public void f() {
        A x = new A();
        B y = new B();
   }
}
```

   (a) Na classe C existem dois objetos, x e y, sendo que x é um objeto da classe B e y da classe A.

   (b) ) O objeto x pode acessar livremente, tanto para leitura como atribuição de valores, o atributo d, uma vez que este atributo é definido na sua classe.

   (c) A classe B não apresenta métodos.

   (d) Existe herança, sendo que B é subclasse de A, portanto todos os atributos existentes em A podem ser utilizados sem qualquer restrição em B.

   (e) Existe um erro no código, pois o atributo d da classe A não pode ser acessado na classe B uma vez que está encapsulado.

9. Considere o trecho de código em **Java**, abaixo.

```
class Gerente extends Funcionario {
     int senha;

     public boolean autentica(int senha) {
         if (this.senha == senha) {
              System.out.println("Acesso Permitido!");
              return true;
         }
         else {
              System.out.println("Acesso Negado!");
              return false;
         }
     }
}
```

   Observando-se os conceitos de orientação a objetos, expostos no trecho em **Java**,

   (a) todo **Funcionario** é um **Gerente**, ou seja, **Gerente** é classe mãe de Funcionario e **Funcionario** é classe filha de **Gerente**.

   (b) a classe **Funcionario** também herda os atributos e métodos privados de **Gerente**, porém não consegue acessá-los diretamente.

   (c) **Gerente** é a superclasse de **Funcionario** e **Funcionario** é a subclasse de **Gerente**.

   (d) a classe **Gerente** herda todos os atributos e métodos da classe **Funcionario**. Isso é expresso pelo uso da palavra chave ***extends***.

   (e) sempre que um objeto do tipo **Funcionario** for criado, este objeto possuirá também os atributos definidos na classe **Gerente**, pois um **Funcionario** é um **Gerente**.