Ayudantías

Programación Orientada a Objetos

Semana 7-Herencia, Interfaces y Clases Abstractas

1. Teoria

1.1 Considera la siguiente definición de clase

```
public final class MtermEx {
        private int x;
        public MtermEx(int newValue){
                x = newValue;
        }
}
```

1.1.1 ¿Cuál de los siguientes enunciados son verdaderos sobre la clase MtermEx?

a. No tiene una clase padre.

b. Su clase padre es Object.

c. No compilara porque una clase no puede ser final.

d. No puede ser extendida.

e. Tiene una subclase por default llamada Object.

1.1.2 SI q1 y q2 son instancias de MtermEx, entonces q1.equals(q2)

a. Es un error de sintaxis ya que el método equals no está definido en la clase MtermEx.

b. Es verdadero si q1 y q2 almacenan el mismo valor en el atributo

c. Es verdadero si q1 y q2 referencian al mismo objeto MtermEx.

d. Nunca es verdadero.

e. lanza una excepción NullPointerException.

1.2 Escriba Verdadero o Falso Según Corresponda

Los constructores de una clase padre se pueden heredar y sobrescribir siempre y cuando no sean privados.	
En un método sobrescrito el nivel de acceso no debe de ser más restrictivo y puede ser menos restrictivo.	
En una una clase abstracta todos los métodos deben ser abstractos.	
En la sobreescritura de métodos se realiza un proceso de enlace dinámico.	
Cuando existe una relación de herencia y se crea un objeto de una subclase S, se ejecuta no sólo el constructor de S sino también el de todas las superclases de S.	

1.3 Seleccione los enunciados correctos respecto a los constructores

a. Un constructor puede devolver cualquier tipo de datos
b. Un constructor público puede ser heredado
c. Si un constructor quiere hacer referencia al constructor de su clase padre debe usar super en la primera línea del constructor
d. Un constructor puede tener cualquier cantidad de argumentos.

1.4 ¿Qué cambios deben hacerse para que este código compile? (Seleccione todas las que aplique)

```
class Hierba{
    protected static String s = "";
    final void crecer() { s += "grow "; }
    static final void crecerRapido() { s += "fast "; }
}
public class Cilantro extends Hierba{
    void crecer() { s += "t-grow "; }
    void crecerRapido() { s+= "t-fast "; }
}
```

a. s debe ser marcado como público.
b. Cilantro.crecer() debe ser marcado como final.
c. Hierba.crecer() NO debe ser marcado como final.
d. Hierba.crecerRapido() NO debe ser marcado como final.
e. Hierba.crecerRapido() NO debe ser marcado como estático.

Justificación:

1.5 Examine la implementación de las clases Filosofo y Kant que se muestran a continuación

```
class Filosofo{
   Filosofo(String s) { System.out.print(s + " "); }
}
public class Kant extends Filosofo{
  // insertar codigo aqui
  public static void main(String[] args) {
        new Kant("Homer");
        new Kant();
  }
}
```

¿Qué fragmentos de códigos a continuación deberían ser insertados en la clase Kant para que al ejecutarla se presente por pantalla **"Homer Bart "**

a. Kant() { this("Bart"); }
   Kant(String s) { super(s); }

b. Kant() { super(); }
   Kant(String s) { super(s); }

c. Kant() { super("Bart"); }
   Kant(String s) { this(); }

d. Kant() { super("Bart"); }
   Kant(String s) { this("Homer"); }

Justificación:

1.6 Examine la implementación de las clases Filosofo y Kant que se muestran a continuación

```
interface Lado { String getLado(); }

class Cara implements Lado {
   public String getLado() { return "Cara "; }
}
class Sello implements Lado {
   public String getLado() { return "Sello "; }
}
class Moneda {
   public static void sobrecarga(Cara side) { System.out.print(side.getLado()); }
   public static void sobrecarga(Sello side) { System.out.print(side.getLado()); }
   public static void sobrecarga(Lado side) { System.out.print("Lado-"+side.getLado()); }
   public static void sobrecarga(Object side) { System.out.print("Object: "); }
   public static void main(String []args) {
      Lado pimerIntento = new Cara();
      Sello segundoIntento = new Sello();
      sobrecarga(pimerIntento);
      sobrecarga((Object)pimerIntento);
      sobrecarga(segundoIntento);
      sobrecarga((Lado)segundoIntento);
    }
}
```

¿Cuál sería la salida del ejecutar la clase Moneda? (Seleccione la respuesta correcta)

A. Cara Cara Sello Sello
B. Sello Object Sello Lado-Sello
C. Cara Object Sello Sello
D. Lado-Cara Object Sello Lado-Sello
E. Error de compilación

Justificación:

1.7 ¿Cuál es la salida en pantalla de ejecutar el siguiente código? - Justifique su respuesta

```
public class Persona {
   String nombre;
   public Persona(String n){ nombre = n;}
   public String toString(){ return nombre; }
   public static void main(String[] args){
      ArrayList<Persona> ar1 = new ArrayList<Persona>();
      ar1.add(new Persona("jose"));
      ar1.add(new Persona("pedro"));
      ArrayList<String> ar2 = new ArrayList<String>();
      ar2.add("juan");
      ar2.add("marcos");
      test1(ar1);
      test2(ar2);
      System.out.println(ar1.get(0)+","+ar1.get(1));
      System.out.println(ar2.get(0)+","+ar2.get(1));
   }
   public static void test1(ArrayList<Persona> arr){
      for(Persona p: arr){ p.nombre =
p.nombre.toUpperCase(); }
   }
   public static void test2(ArrayList<String> arr){
      for(String s: arr){ s = s.toUpperCase(); }
   }
}
```

Salida:

Justificación:

2. verificación de Código

```
interface GFG {
	void myMethod();
	void getInfo();
}

abstract class Geeks implements GFG
{
	void getData(){
		System.out.println("GFG");
	}
}

public class Test extends Geeks
{
	public void myMethod(String message){
		System.out.println(message);
	}
	public void getInfo(){
		System.out.println("Geeks");
	}
	public static void main(String[] args){
		Geeks obj = new Test();
		obj.myMethod("GeeksforGeeks");
	}
}
```

¿Cuál sería la salida?

a. Error de ejecución
b. GeeksforGeeks
c. Error de compilación
d. Ninguna de las anteriores

Justifique:

```
class C {
      void p(C c) {
        System.out.print("AB ");
      }
      void p(D d) {
        System.out.print("CD ");
      }
}

class D extends C {
      void p(C c) {
         System.out.print("DE ");
      }
}

public class InvocacionMetodos {
      public static void main(String[] args){
         C obj = new D();
         obj.p(obj);
         ((D) obj).p(obj);
         obj.p((D) obj);
      }
}
```

¿Cuál sería la salida?

a. AB DE CD
b. DE DE CD
c. AB AB CD
d. Error de compilación
e. Ninguna de las anteriores

Justifique:

```
class Fizz {
    int x = 5;
    public static void main(String[] args) {
        final Fizz f1 = new Fizz();
        Fizz f2 = new Fizz();
        Fizz f3 = FizzSwitch(f1,f2);
        System.out.print((f1 == f3) + " ");
        System.out.print((f1.x == f3.x));
    }
    static Fizz FizzSwitch(Fizz x, Fizz y) {
        Fizz z = x;
        z.x = 6;
        return z;
   }
}
```

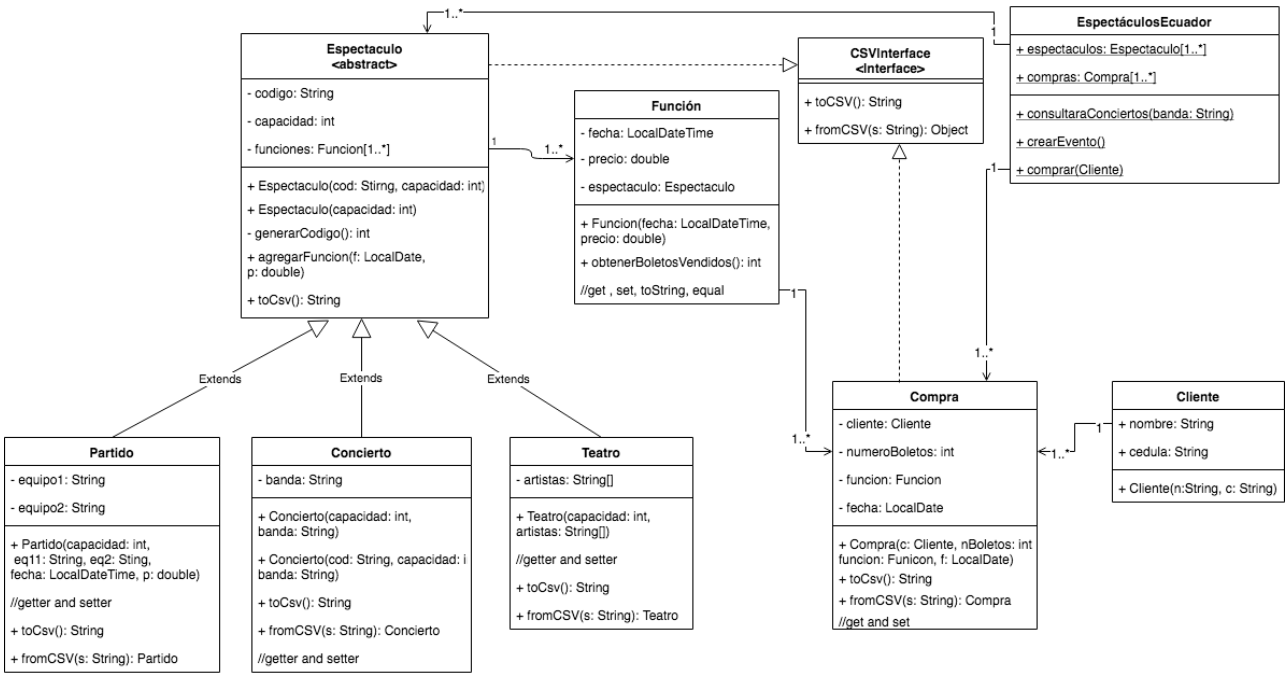
¿Cuál sería la salida?

a. true true
b. false true
c. true false
d. false false
e. Error de compilación

Justifique:

3. Desarrollo

Una empresa organizadora de espectáculos lo contrata a usted para que cree un sistema donde los clientes puedan adquirir entradas para los espectáculos de su interés. Para ello la empresa le proporciona el siguiente diagrama de clases con el diseño del sistema.



En base al diagrama de clases anterior, realice lo siguiente:

1. Escriba el código de la interfaz **CSVInterface**
2. Implemente la clase **Espectaculo.** Tenga en cuenta lo siguiente:
    - El método **generarCódigo** ya ha sido implementado y retorna un código único para el espectáculo (**NO HACERLO**)
    - **Solo** debe implementar los métodos get y set del atributo **capacidad.**
    - El método **agregarFuncion** crea una nueva función y la agrega a la lista de funciones del Espectáculo.
    - El método **toCSV** retorna un String con los datos del Espectaculo separados por comas.

"cod, capacidad, fechaFuncion1| precio1; fechaFuncion2| precio2"

3. ¿Por qué la clase Espectáculo debe ser declarada abstracta? - Justifique su respuesta.
4. Implemente la clase **Concierto**. Tenga en cuenta lo siguiente:

- **NO IMPLEMENTE** los métodos gets and sets de la clase. Considere que estos ya han sido implementados.
- El método **toCSV** retorna un String con los datos del **Concierto** separados por comas.

"cod, capacidad, fechaFuncion1| precio1; fechaFuncion2| precio2, banda"

- El método **fromCSV** recibe un String del mismo formato que el devuelto por la función anterior y retorna un objeto de tipo Concierto

5. En la clase Funcion haga lo siguiente:
   - Implemente el método **obtenerBoletosVendidos** que retorna el número de boletos que han sido vendidos para esa función.
6. En la clase **EspectáculosEcuador** haga lo siguiente:

Implemente el método **consultarConciertos que recibe por parámetro el nombre de una banda** y muestra en pantalla todos los conciertos de esa banda. De cada concierto se debe mostrar la siguiente información:

(Considere que el método toString() está sobrescrito en la clase Funcion y devuelve un String con el formato mostrado abajo).

```
Código: 2345
Capacidad: 234
Banda: Linkin Park
Funciones:
    21/Diciembre/2018 22:00, Precio: 80, Boletos Disponibles: 10
    22/Diciembre/2018 23:00, Precio: 80, Boletos Disponibles: 12
    23/Diciembre/2018 23:00, Precio: 80, Boletos Disponibles: 15
```