Programación orientada a objetos. Primer parcial 2003.

Viernes 2 de mayo de 2003.

1. Sean las clases `Persona` y `Cliente` que aparecen a continuación.

```
public class Persona {
  …
  public String getEdad(){…}
  public String getNombre(){…}
  public void setEdad(int edad){…}
  public void setNombre(String nombre){…}
}

public class Cliente extends Persona {
  …
  public String getNombre(boolean incluyeApellido){…}
  public void setNombre(String nombre) {…}
}
```

   a. El método `getNombre()` implementado clase `Cliente` ¿sobrescribe al método `getNombre()` de la clase `Persona`? Justifiquen brevemente la respuesta.

   b. ¿Cuántos métodos tiene la clase `Cliente`? Justifiquen brevemente la respuesta.

   c. Enumeren los tipos que tienen los objetos de la clase `Cliente`.

   Ignoren la clase `Object` en las respuestas.

15 puntos

2. Asuman que en el ejercicio anterior no es posible usar herencia. Definan nuevamente la clase `Cliente` para que tenga el mismo tipo.

10 puntos

3. Supongan que existen dos clases diferentes que no tienen métodos con signaturas en común. ¿Es posible tener un objeto capaz de ejecutar todos los métodos de esas clases? ¿Cómo? ¿Hay más de una manera? ¿Cuáles? No pueden escribir los métodos; deben usar los que ya están implementados.

   Además de contestar las preguntas anteriores programen las respuestas en Java usando las siguientes clases:

```
public class Cliente {
  public String getNombre() {…}
  public void setNombre(String nombre) {…}
}

public class Contribuyente {
  public int getRUC() {…}
  public void setRUC(int RUC) {…}
}
```

15 puntos

4. Un objeto puede tener más de un tipo y varios objetos de diferentes clases pueden tener el mismo tipo. Una interfaz define un tipo y una clase también define un tipo. Muestren con un pequeño fragmento de código Java un ejemplo de:

   a. Un objeto con más de un tipo definiendo los tipos mediante interfaces.

   b. Un objeto con más de un tipo definiendo los tipos mediante clases.

   c. Varios objetos de diferentes clases y el mismo tipo definiendo los tipos mediante interfaces.

   d. Varios objetos de diferentes clases y el mismo tipo definiendo los tipos mediante clases.

10 puntos

5. ¿Qué es una clase abstracta? Supongan un lenguaje en el que no es posible definir interfaces pero sí es posible definir clases abstractas ¿Es posible hacer en ese lenguaje todo lo mismo que hacemos en Java con las interfaces? Justifiquen la respuesta.

10 puntos

6. Programen en Java una clase `StringHandler` con los métodos necesarios para que se pueda hacer lo siguiente con un objeto de la clase `String`:

   a. Obtener la cadena al revés: `String alReves(String s)`.

   b. Obtener la cantidad de vocales en la cadena: `int contarVocales(String s)`.

   c. Obtener la cantidad de ocurrencias de una letra dada en la cadena: `int contarLetra(String s, Char c)`.

10 puntos

7. Definan en forma breve y precisa qué es el estado y qué es el comportamiento de un objeto. Hagan un pequeño programa en Java en el que haya un objeto sin estado y otro en el que haya un objeto sin comportamiento.

10 puntos

8. Hagan un programa `Prueba` en Java que reciba como argumento una cadena y, usando la clase `StringHandler` del ejercicio anterior, imprima en la consola la cadena al revés y luego termine; es decir si se ejecuta el comando

   ```
   java Prueba "Hola que tal"
   ```

   se verá en la consola

   ```
   lat euq aloH.
   ```

   Además del código fuente del programa deberán indicar el nombre del archivo que debe contener ese código.

10 puntos

9. Sea una interfaz `Serializable` y una clase `Serializer` definidas así:

```
public interface Serializable {
  public void serializeTo(Stream s);
}

public class Serializer {
  public void serialize(Object o, Stream s){…}
  public void serializeTo(Stream s) {
    serialize(this, s);
  }
}
```

   La operación `serializeTo(Stream s)` guarda el receptor en la `Stream s`. El método `serialize(Object o, Stream s)` guarda el `Object o` en la `Stream s`. Se pide a dos programadores que implementen una nueva clase con un método que guarde el receptor en una `Stream`. Un programador propone la clase `MySerializerObject` y otro la clase `MySerializableObject` declaradas así:

```
class MySerializerObject extends Serializer {
}

class MySerializableObject implements Serializable {
  private Serializer p = new Serializer();

  public void serializeTo(Stream s) {
    p.serialize(this, s);
  }
}
```

   ¿Qué usó cada programador? Comparen ambas soluciones indicando ventajas y desventajas

10 puntos