Laboratorio de Software Práctica nº 1

Temas

- Especificadores de acceso: private, public, protected, default.
- Constructores de Clases.
- Clases abstractas.
- **1.-** Escriba una clase llamada **Estudiante** con 3 variables de instancia: apellido, nombre y legajo. Implemente los getters y setters para cada una de las variables de instancias anteriores.
 - a) Sobre-escriba el método **toString()** de **Object,** para ello declare una variable local de tipo **StringBuffer** y utilícela para concatenar cada uno de los datos del estudiante y retorne un objeto **String** con los datos del mismo.
 - b) Escriba el método **main()** en la clase **TestEstudiante**, donde se debe crear un arreglo con 5 objetos **Estudiante** inicializados, para luego recorrer el arreglo e imprimir en pantalla los objetos guardados en él.
 - c) Comente el método **toString()** escrito en la clase **Estudiante** y vuelva a ejecutar el programa. ¿Cuál es la diferencia entre b) y c)?
 - d) Cree otro objeto de tipo Estudiante y compárelo con el anterior. ¿Qué método de **Object** es utilizado para la comparación por contenido?.
 - e) Ejecute la aplicación fuera del entorno de desarrollo. ¿Para que se utiliza la variable de entorno **CLASSPATH**?
 - f) Construya un archivo jar con las clases anteriores, ejecútelo desde la línea de comandos. ¿Dónde se especifica en el archivo jar la clase que contiene el método main?
- **2.-** Analice las siguientes clases y responda cada uno de los incisos que figuran a continuación.
- a) Considere la siguiente clase Alpha. ¿Es válido el acceso de la clase Gamma?. Justifique.

```
package griego;
class Alpha {
         protected int x;
         protected void otroMetodoA() {
             System.out.println("Un método protegido");
        }
}

package griego;
class Gamma {
        void unMétodoG() {
             Alpha a = new Alpha();
             a.x=10;
             a.otroMetodoA();
        }
}
```

b) Considere la siguiente modificación de la clase **Alpha**. Son válidos los accesos en la clase **Beta**?. Justifique.

```
package griego;
public class Alpha {

public int x;
public void unMetodoA() {
   System.out.println("Un Método Público");
}

package romano;
import griego.*;
class Beta {
   void unMetodoB() {
        Alpha a=new Alpha();
        a.x=10;
        a.unMetodoA();
   }
}
```

c) Modifique la clase **Alpha** como se indica debajo. ¿Es válido el método de la clase **Beta**?. Justifique.

```
package griego;
public class Alpha {
 int x;
 void unMetodoA(){
  System.out.println("Un mét. paquete");
 }
}
package romano;
import griego.*;
class Beta {
      void unMetodoB(){
             Alpha a = new Alpha();
              a.x=10;
              a.unMetodoA();
        }
   }
```

- d) Considere el inciso c) $\dot{\epsilon}$ Es válido el acceso a la variable de instancia x y al método de instancia unMetodoA() desde una subclase de **Alpha** perteneciente al paquete **romano**?. Justifique.
- e) Analice el método de la clase **Delta**. ¿Es válido? Justifique analizando cómo influye el control de acceso **protected** en la herencia de clases.

- 3.- Respecto de los constructores, analice los siguientes casos:
- 3.1.- Escriba 3 subclases de la clase **Estudiante** (definida en el punto **1**) llamadas **EstudianteUniversitario**, **EstudianteSecundario** y **EstudiantePrimario** con las siguientes variables de instancias:
 - **EstudianteUniversitario**: define una variable de instancia destinada para la fecha de ingreso a la institución.
 - EstudianteSecundario: define 2 variables de instancia, una para guardar la cantidad de materias previas y la otra para mantener registro del promedio general del alumno.
 - **EstudiantePrimario**: define 2 variables de instancia, una para mantener la cantidad de ausencias y la otra para saber si alguna vez repitió de grado.
 - a) Implemente los getters y setters para cada una de las variables de instancias anteriores.
 - b) Implemente los constructores para las clases anteriores, todos ellos deben recibir los parámetros necesarios para inicializar las variables de instancia propias de la clase donde están definidos.
 - c) ¿Pudo compilar las clases? ¿Qué problemas surgieron y por qué? ¿Cómo los solucionó?
- 3.2.- El siguiente código, define una subclase de **java.awt.Dialog**. Verifique si compila. Si no lo hace implemente una solución.

```
package laboratorio;
import java.awt.Dialog;
public class Dialoguito extends Dialog {
   public Dialoguito() {
       System.out.println("Dialoguito");
   }
}
```

Nota: Recuerde que en la url http://docs.oracle.com/javase/8/docs/api/ tiene disponible al documentación de la API de java

3.3.- Las clases definidas a continuación establecen una relación de herencia. La implementación dada, ¿es correcta?.

Constructores privados

```
package laboratorio;
public class SuperClase {
   private SuperClase() {
   }
}

package laboratorio;
public class SubClase extends SuperClase {
      public SubClase() {
   }
}
```

Constructores protegidos

```
package laboratorio;
public class SuperClase{
  protected SuperClase() {
  }
}

package laboratorio1;

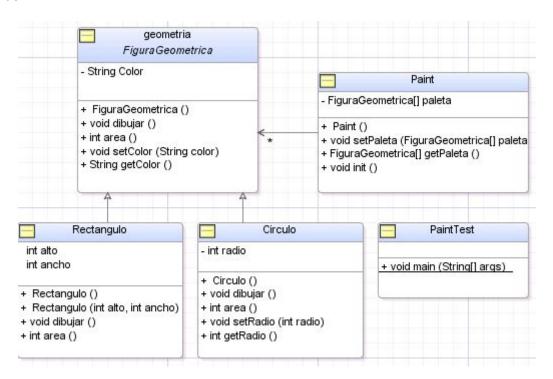
public class SubClase extends SuperClase {
    public SubClase() {
    }
}

package laboratorio1;

public class OtraClase {
    public OtraClase() {
    }
    public void getX() {
        new SuperClase();
    }
}
```

5.- Hay un número de casos donde se necesitan tener una <u>única instancia por clase</u>, esto es lo que se conoce como el patrón <u>Singleton</u>. Implemente una clase que cumpla con este patrón llamada **PoolConexiones**, la misma debe proveer una manera para acceder a esa instancia. Piense en los modificadores de acceso del constructor y en los calificadores java que tiene disponibles, para buscar una solución.

6.- Escriba las siguientes clases java que figuran en el siguiente diagrama UML **respetando** cada una de las especificaciones para las clases y las relaciones entre ellas:



Tenga en cuenta lo siguiente:

- La clase FiguraGeometrica es una <u>clase abstracta</u> con 2 métodos abstractos dibujar() y area() y el resto de los métodos concretos.
- Las subclases Rectangulo y Circulo son clases concretas. Ambas deben implementar el método dibujar() simplemente imprimiendo un mensaje en la consola. Por ejemplo: "se dibuja un círculo de radio 2 y de color azul" (donde el radio y el color son variables de instancia). El método getArea() debe implementarse en cada subclase de FiguraGeometrica.
- En la clase Paint, el método init() debe crear las instancias de Rectangulo y Circulo y guardarlas en el arreglo paleta. Los valores para crear estas instancias son los siguientes:
 - o Defina 2 objetos Circulo (radio 2 y color azul, radio 3 y color amarillo)
 - Defina 3 objetos Rectangulo (alto 2, ancho 3, color verde y alto 4 y ancho 10 y color rojo).
- La clase PaintTest debe crear una instancia de Paint, inicializarla y recorrerla. En cada iteración invoque el método area() sobre el elemento actual y getRadio(), sólo si se trata de un objeto de tipo Circulo.