Laboratorio de Software
Práctica nº 1

Temas
- Especificadores de acceso: private, public, protected, default.
- Constructores de Clases.
- Clases abstractas.

1.- Escriba una clase llamada **Vacuna** con 4 variables de instancia: marca, país de origen, enfermedad que previene y cantidad de dosis. 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 de la vacuna y retorne un objeto **String** con los datos del mismo.
b) Escriba el método **main()** en la clase **TestVacuna**, donde se debe crear un arreglo con 5 objetos **Vacuna** inicializados, para luego recorrer el arreglo e imprimir en pantalla los objetos guardados en él.
c) Comente el método **toString()** escrito en la clase **Vacuna** y vuelva a ejecutar el programa. ¿Cuál es la diferencia entre b) y c)?
d) Cree otro objeto de tipo Vacuna 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) ¿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.

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

package romano;
import griego.*;
public class Delta extends Alpha {
   void unMetodoD(Alpha a, Delta d){
          a.x=10;
          d.x=10;
          a.otroMetodoA();
          d.otroMetodoA();
         }
}
```

3.- Respecto de los **constructores**, analice los siguientes casos:

3.1.- Escriba 3 subclases de la clase **Vacuna**(definida en el punto **1**) llamadas **VacunaPatogenoIntegro**, **VacunaSubunidadAntigenica** y **VacunaGenetica** con las siguientes variables de instancias:

- **VacunaPatogenoIntegro**: define una variable de instancia destinada para el nombre del virus patógeno inactivado o atenuado.
- **VacunaSubunidadAntigenica**: define 2 variables de instancia, una para guardar la cantidad de antígenos de la vacuna y la otra para mantener el tipo de proceso llevado a cabo.
- **VacunaGenetica**: define dos variables de instancia, una para la temperatura mínima y otra para la temperatura máxima de almacenamiento.

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.io.File.** Verifique si compila. Si no lo hace implemente una solución.

```
package laboratorio;
import java.io.File;
public class MiArchivo extends File {
    public MiArchivo() {
        System.out.println("Mi Archivo instanciado") ;
    }
}
```

Nota: Recuerde que en la url https://docs.oracle.com/en/java/javase/24/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

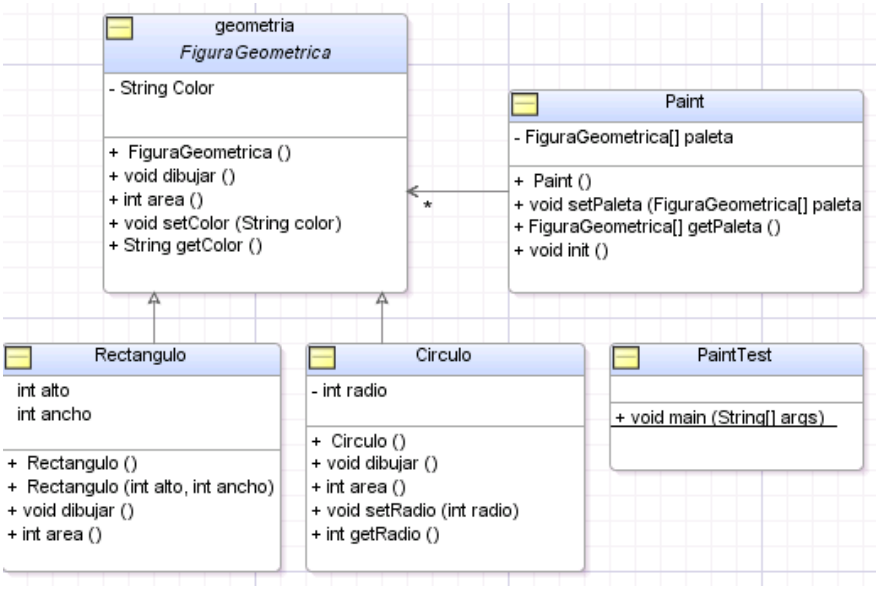
```
package laboratorio;
public class SuperClase {
  private SuperClase() {
  }
}

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();
  }
}
```

4.- Implemente una clase llamada **Logger** que siga el patrón Singleton. Esta clase debe garantizar que solo haya una instancia de Logger a lo largo de la ejecución de la aplicación. Proporcione un método estático getInstance() para acceder a esta instancia. Además, debe incluir métodos para registrar mensajes de diferentes tipos, como logInfo(String mensaje), logWarning(String mensaje) y logError(String mensaje), que simulen el registro de mensajes mostrándolos en la consola. Piense en los modificadores de acceso del constructor, para buscar una solución.

5.- 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 **clase abstracta** 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 **`area()`** 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:
  - Defina 2 objetos Circulo (radio **2** y color **azul**, radio **3** y color **amarillo**)
  - Defina 2 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`**.
- Construya un archivo jar ejecutable con las clases anteriores. El mismo debe poderse ejecutar como un programa haciendo doble click.