Practica 1

Ejercicio 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)?

El resultado de imprimir el objeto Vacuna sin sobrescribir método toString() es el siguiente:

```
practica1.ej1.Vacuna@7daf6ecc
practica1.ej1.Vacuna@2e5d6d97
practica1.ej1.Vacuna@238e0d81
practica1.ej1.Vacuna@31221be2
practica1.ej1.Vacuna@377dca04
```

La diferencia es que en el punto b) el resultado de la impresión será el que nosotros estipulamos en el método toString() mientras que en el punto c) se mostrará una representación en cadena del objeto. Esta esa conformada por:

- El paquete donde está situado la clase de donde proviene el objeto.
- El nombre de la clase.
- La representación del objeto como un numero hexadecimal (este es único a ese objeto).

d) Cree otro objeto de tipo Vacuna y compárelo con el anterior. ¿Qué método de Object es utilizado para la comparación por contenido?.

El método que se utiliza para comparar dos objetos en cuanto a su contenido es el Object.equals(). Este tiene que ser sobrescrito para que determine si dos objetos, el primero que lo llama y el segundo que es pasado por argumento, son iguales comparando sus variables.

e) Ejecute la aplicación fuera del entorno de desarrollo. ¿Para qué se utiliza la variable de entorno CLASSPATH?

La variable de entorno CLASSPATH que contiene la lista de directorios usados como raíces para buscar los archivos .class. El JVM toma el nombre del paquete de la sentencia import y reemplaza cada "." por una barra "\" 0 "/" (según el SO) para generar un path a partir de las entradas del 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?

Se especifica en el archivo MANIFEST.MF que está en META-INF/MANIFEST.MF. Este indica cómo se usa el archivo JAR. Las aplicaciones de escritorio a diferencia de las librerías de componentes o utilitarias requieren que el archivo MANIFEST.MF contenga una entrada con el nombre de la clase que actuará como punto de entrada de la aplicación, la que define el método main().

Ejercicio 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();
	}
}
```

Si, es válido el acceso de la clase Gamma ya que la variable y el método a los que intenta acceder (sobre Alpha) son de acceso *protected*, es decir, el autor de la clase base podría determinar qué miembros pueden ser accedidos por las subclases, pero no por todo el mundo. Esto es protected. Además, el acceso protected provee acceso package: las clases declaradas en el mismo paquete que el miembro protected tienen acceso a dicho miembro.

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();
	}
}
```

Si, es válido mientras se haga import de los nombres que se utilizan, ya que el atributo, método o constructor declarado public está disponible para todos.

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();
	}
}
```

No, es invalido ya que si no se especifica el tipo de acceso este package o friendly o privado del paquete. Esto Implica que tienen acceso a dicho miembro solamente las clases ubicadas en el mismo paquete que él. Para las clases declaradas en otro paquete, es un miembro privado.

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.

No, es invalido ya que como anteriormente se menciono el tipo de acceso package (default) da permisos solo a aquellas clases que están en el mismo paquete. Utilizar herencia no cambia este comportamiento.

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();
	}
}
```

No, es invalido ya que Delta intenta acceder a los atributos de Alpha a través de su interfaz, la cual es protegida. Estos elementos solo pueden ser accedidos por las clases que heredan de Alpha. Si bien Delta es una subclase de Alpha no esta accediendo a sus elementos heredados (this) sino que está queriendo conocer los datos de una instancia de la clase Alpha.

Ejercicio 3.

Respecto de los constructores, analice los siguientes casos:

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?

Si se pueden compilar, pero el problema que surgirá es que los atributos base de la superclase se inicializaran con sus valores default. Esto dado a que no los especificamos al momento de construir la clase hija. Para corregir esto se debe crear un constructor en la clase Vacuna, donde se mande por parámetros los valores de los atributos. Este constructor será llamado en la subclase a través del método super(), con sus respectivos argumentos.

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") ;
      }
}
```

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();
	}
}
```

Para los constructores privados la implementación es incorrecta ya que estos no son conocidos por sus subclases, impidiendo que estas se instancien de forma default.

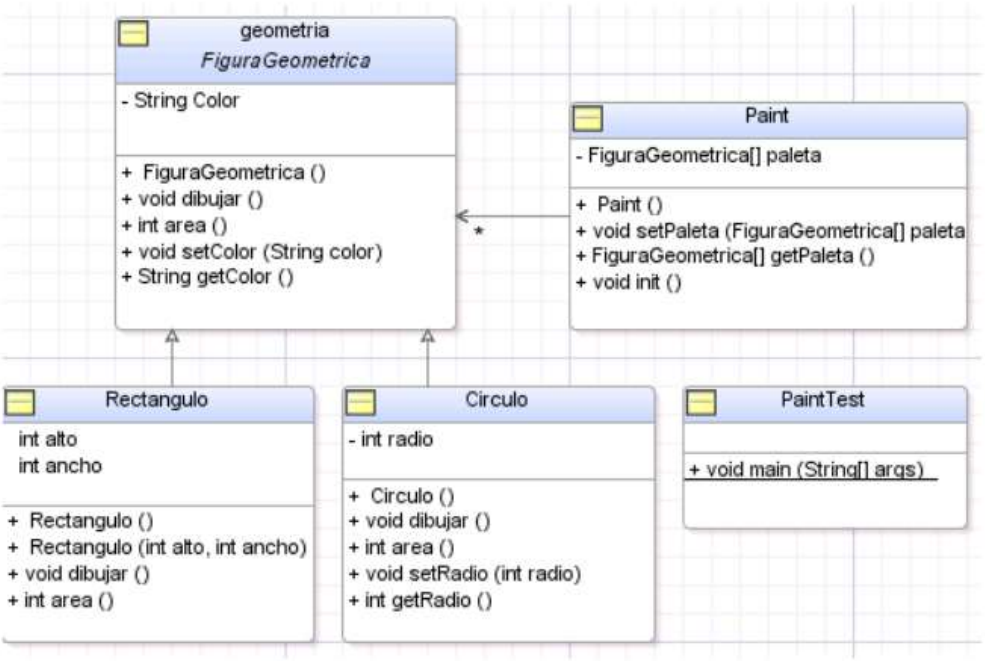
Para los constructores protegidos la implementación es incorrecta ya que la clase que intenta instanciar a SuperClase (OtraClase) está en otro paquete, y el método constructor solo es conocido por las subclases y otras clases del mismo paquete.

Ejercicio 5.

Hay un número de casos donde se necesita tener una única instancia por clase, esto es lo que se conoce como el patrón Singleton. 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.

Ejercicio 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 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.