Laboratorio de Software
Práctica nº 5

Temas
- Conceptos y uso de anotaciones
- Definición de anotaciones

1.- Analice qué ocurre con la siguiente clase cuando se compila:

```
public class TestSobreescritura {
	@Override
	public String tostring() {
		return super.toString() + " Testeando: 'Override'";
	}
}
```

a) ¿Qué ocurre cuando se ejecuta **TestAnotaciones**?
b) ¿Qué ocurre si se elimina @SuppressWarnings({"deprecation"})? ¿el resultado de la ejecución es el mismo?
c) ¿Cuál es la diferencia entre anotar el método **testarYa()** y anotar la clase **TestAnotaciones**?

```
public class TestDeprecated {
	@Deprecated
	public void hacer() {
		System.out.println("Testeando: 'Deprecated'");
	}
}

public class TestAnotaciones {
   public static void main(String arg[]) throws Exception {
      new TestAnotaciones().testearYa();
   }
   @SuppressWarnings({"deprecation"})
   public void testearYa() {
      TestDeprecated t2 = new TestDeprecated();
      t2.hacer();
   }
}
```

2.- Implementar una clase que mapee un objeto Bean a un archivo del filesystem y almacene en el archivo:
1. El nombre de la clase.
2. Los nombres de los atributos y el contenido de los mismos.

Las anotaciones que entiende son las siguientes:

- `@Archivo(name="nombre.extension")` -- Indica que la información se almacenará en el archivo nombre.extension. Si no se explicita un nombre se utiliza el nombre de la clase.
- `@AlmacenarAtributo` -– Denota que el nombre del atributo que está a continuación de la anotación se debe almacenar en el archivo.

Por ejemplo:

Dada la siguiente clase:

```
@Archivo(nombre="AchivoMapeado.txt")
public class Mapeado {
	@AlmacenarAtributo
	private String valor = "Default1";

	@AlmacenarAtributo
	private Integer valor2=20;

	@AlmacenarAtributo
	private Float valor3=30.20f;

	private Float valor4=30.20f;

	//Metodos getters y setters
}
```

La salida en el archivo **`AchivoMapeado.txt`** sería:

```
<nombreClase>Mapeado</nombreClase>
<nombreAtributo>valor</nombreAtributo>
<nombreValor>Default1</nombrenombreValor>
<nombreAtributo>valor2</nombreAtributo>
<nombreValor>20</nombrenombreValor>
<nombreAtributo>valor3</nombreAtributo>
<nombreValor>30.2</nombrenombreValor>
```

3.- Implementar el siguiente ejercicio:

a. Definir la anotación `RUNTIME` llamada **`@Servidor`** que se utiliza para anotar una clase que funcionará como un servidor HTTP. Esta anotación debe poseer los siguientes atributos:
   - **`dirección`** – indica la dirección IP a la cual se conectarán los clientes.
   - **`puerto`** – indica el puerto donde espera las conexiones de los clientes.
   - **`archivo`** – indica el archivo en el que se guardará la información de login.

b. Definir la anotación `RUNTIME` llamada **`@Invocar`** que se utiliza para marcar el o los métodos de clase que deben ser invocados cuando un cliente se conecta al servidor.

c. Utilice las anotaciones previamente definidas para anotar una clase cualquiera.

d. Implementar una clase llamada **Contenedor** que procese la clase anotada para escuchar peticiones de red en la IP y puerto especificados y luego delegar la atención de las mismas en los métodos de la clase anotada. La clase **Contenedor** al recibir una petición deberá realizar dos tareas:

   - Loguear Fecha, Hora e IP del cliente en un archivo de texto cuyo nombre se indicó en la anotación **@Servidor**

   - Invocar a todos los métodos que fueron anotados con la anotación **@Invocar**

e. Pruebe el servidor HTTP creado en los incisos anteriores con un navegador de Internet.