TALLER No. 8	PROGRAMACIÒN ORIENTADA A OBJETOS

Profesor: Jaime Alberto Guzmán Luna

Contenido del taller:

1. Interfaces.

1. Ejercicio de clase

```
1  import java.util.*;
2  public class ObjTaller8 {
3      public static void main (String[] args) {
4
5          //Defina el operador diamante del siguiente arreglo.
6          ArrayList<???????> listaVehiculos = new ArrayList<???????>();
7
8          for (int i = 0; i < listaVehiculos.size(); i++) {
9              //Obtiene cada elemento en el arreglo y lo almacena
10             //en la variable v.
11             ??????? v =  listaVehiculos.get(i);
12         }
13       }
14 }
15
16 interface Emergencia {
17     int VOLUMEN = 10;
18     String sonarSirena();
19     int getVolumen();
20 }
21
22 interface Vehiculo {
23     String getNombre();
24     int getMaxPasajeros();
25     int getMaxVelocidad();
26 }
27
28 interface VehiculoTierra extends Vehiculo {
29     int getNumeroRuedas();
30     void agregarRuedas(int numeroRuedas);
31     String conducir();
32 }
33
34 interface VehiculoAgua extends Vehiculo {
35     String soltarAmarras();
36 }
37
38 class Sedan implements VehiculoTierra {
```

```
39        private String nombre;
40        private short maxPasajeros;
41        private int maxVelocidad;
42        private int numeroRuedas;
43
44        public Sedan() {
45            this("Mi Sedan", (short) 4, 200);
46        }
47
48        public Sedan(String nombre, short maxPasajeros, int maxVelocidad)
    {
49          this.nombre = nombre;
50          this.maxPasajeros = maxPasajeros;
51          this.maxVelocidad = maxVelocidad;
52            this.numeroRuedas = 0;
53        }
54
55        public void agregarRuedas(int numeroRuedas) {
56            this.numeroRuedas = numeroRuedas;
57        }
58
59        public String getNombre() {
60            return this.nombre;
61        }
62
63        public int getMaxPasajeros() {
64            return this.maxPasajeros;
65        }
66
67        public int getMaxVelocidad() {
68            return this.maxVelocidad;
69        }
70
71        public int getNumeroRuedas() {
72            return this.numeroRuedas;
73        }
74
75        public String conducir() {
76            return "Conduce al vehículo: " + nombre;
77        }
78
79        String sonarClaxon() {
80            return "Claxon sonando...";
81        }
82
83        public String toString() {
84            return super.toString();
85        }
86    }
```

Teniendo en cuenta el código anterior, realice lo siguiente:

1. Implemente una nueva clase llamada `Aerodeslizador` Esta nueva clase debe implementar las interface de `VehiculoTierra` y `VehiculoAgua` con las siguientes condiciones:

    a. Defina los atributos necesarios para implementar la clase de acuerdo con la lógica de los métodos que hereda esta clase. Estos atributos deben quedar con el mayor nivel de privacidad para la clase. Tome como ejemplo la clase *Sedan.* Defina un constructor por defecto que inicialice los atributos con valores predefinidos por usted.
    b. Defina otro constructor que inicialice los atributos de esta clase con los valores pasados como parámetros a este constructor.
    c. Implemente el cuerpo de los métodos que hereda esta clase teniendo en cuenta que no se deben cambiar los métodos abstractos declarados en las interfaces. Ninguna lógica de estos métodos debe ejecutar la instrucción `System.out.println.` Esta instrucción será exclusiva del método `main` de la clase `ObjTaller8.`

2. Implemente una nueva clase llamada `Fragata` Esta nueva clase debe heredar de `VehiculoAgua`, con las mismas condiciones indicadas en el numeral anterior.

3. Implemente una nueva clase llamada `PatrullaPolicia`. Esta nueva clase debe heredar de `Sedan` y `Emergencia`. Con las siguientes condiciones:

    a. Implemente un constructor que inicialice los atributos heredados por la superclase.
    b. Implemente los atributos de acuerdo a la lógica de los métodos que está heredando esta clase.
    c. Implemente el cuerpo de los métodos abstractos que está heredando esta clase. Tenga en cuenta que el método `getVolumen()` deben retornar el valor definido por el atributo ***VOLUMEN*** en la interface `Emergencia`. Tenga en cuenta las demás aclaraciones en la condición (c) del numeral (1).
    d. Implemente el siguiente método dentro de la clase `PatrullaPolicia`:

```
public void setVolumen() {
        Emergencia.VOLUMEN++;
}
```

    Explique, ¿Porque esta implementación genera error?

    e. Defina el método `toString()` dentro de la clase `PatrullaPolicia` y **responda:** ¿Se está sobrescribiendo el método? ¿de quién se está sobrescribiendo el método `toString()` en este caso?

4. En el método `main` de la clase `ObjTaller8` realice lo siguiente:

    a. Crear las instancias (objetos) de las clases definidas anteriormente de la siguiente manera.

```
Vehiculo s = new Sedan();
Vehiculo a = new Aerodeslizador();
```

```
Vehiculo f = new Fragata();
Vehiculo p = new PatrullaPolicia("Patrulla 001", 5, 200);
```

b. Agregue 4 ruedas a los vehículos de tierra (al sedan y a la patrulla de policía) y 2 ruedas al aerodeslizador, utilizando el método `agregarRuedas()`. Debe conservar `Vehiculo` como el tipo de los objetos creados en el literal anterior. **NO PUEDE** cambiar el tipo de los objetos creados anteriormente. Para este punto tenga en cuenta el casting de objetos.

c. Cree el siguiente arreglo:

```
ArrayList<???????> listaVehiculos = new ArrayList<???????>();
```

Responda: ¿Qué debe ir en el operador diamante del ArrayList?

d. Agregue **TODOS** los objetos creados en el literal (a) de este numeral al ArrayList con el método `add()` de la clase `ArrayList`. P. Ej.: `listaVehiculos.add(s);`

e. Recorra el arreglo creado con el siguiente ciclo:

```
for (int i = 0; i < listaVehiculos.size(); i++) {
  //Obtiene cada elemento en el arreglo y lo
  almacena //en la variable v.
     v =  listaVehiculos.get(i);

     //Implemente nuevo código a partir de esta línea
  }
```

Responda: ¿De qué tipo debe ser la variable *v* en el código anterior?

f. Dentro del ciclo del literal anterior debe mostrar la información que devuelven **TODOS** los métodos que retornan algún valor de cada objeto. Debe conservar `Vehiculo` como el tipo de los objetos creados en el literal (a) del numeral (4). **NO PUEDE** cambiar el tipo del objeto creado anteriormente. Para este punto tenga en cuenta el operador `instanceof` y los respectivos cast. Por ejemplo:

```
System.out.println("Nombre = " + v.getNombre());
System.out.println("Max Pasajeros = " + v.getMaxPasajeros());
System.out.println("Max Velocidad = " + v.getMaxVelocidad());
.
.
.
//Información del resto de métodos definidos para cada clase.
```

g. Si se quisiera subir el volumen de la sirena de la patrulla de policía usando un método `setVolumen()`. **Responda:** ¿Qué debería hacerse y por qué?

h. La línea 84 está haciendo un llamado al método `toString()`. ¿En dónde está definido este método para su invocación?

2. Ejercicio de análisis

Dado el siguiente código, ¿cuál es el resultado de ejecutar la clase `ObjTaller8B`? Explique su respuesta.

```
1  interface A {
2        public static int i = 3; void m();
3        void m1();
4  }
5  public class ObjTaller8B implements A {
6
7        public void m() {
8              System.out.println(i);
9        }
10       public void m1() {
11             System.out.println(i + 3);
12       }
13       public static void main(String[] args) {
14             A obj = new ObjTaller8B();
15             obj.m();
16             ObjTaller8B.i = 4;
17             obj.m1();
18       }
19 }
```

3. Ejercicio de análisis

¿Cuál es el resultado de compilar el siguiente código? Explique su respuesta.

```
1  public interface A {
2      void m();
3  }
4
5  interface B extends A {
6      public int m1();
7  }
8
9  abstract class C implements A, B {
10     public int m1() { return 1; }
11 }
12
13 abstract class D extends C implements B {
14     public int m1() { return 0;}
15 }
```

a. ¿Qué sucede en la línea 14? Explique su respuesta.
b. ¿Qué sucede si el método de la línea 2 se vuelve static?

4. EJERCICIO PRACTICO GITHUB

Ingrese al siguiente enlace para aceptar y empezar el ejercicio: https://classroom.github.com/a/nyA1zR3y

4) En este problema se implementará la clase ***Futbolista*** y sus clases derivadas, ***Portero*** y ***Jugador*** Para ello, se pide lo siguiente:

a) Crear la clase Futbolista que tiene las siguientes características:

- Deberá implementar la interface comparable (Nota: La interface comparable se dejó como consulta en su momento).
- Tendrá los siguientes atributos privados: nombre de tipo String, edad de tipo int, y posicion de tipo String. posicion no debe poder cambiar de valor una vez establecida.
- Debe tener un constructor para dar valor a los tres atributos anteriores.
- Debe tener un constructor por defecto, que usando el anterior, cree un Futbolista de nombre "Maradona", posición "delantero" y edad 30 años.
- Añadir el código necesario a la clase para que cuando en una clase usuaria se haga System.out.println(elFutbolista) siendo elFutbolista un objeto de la clase Futbolista, se imprima por pantalla lo siguiente: "El futbolista "+<nombre>+" tiene "+<edad>+ ", y juega de "+<posicion>. Donde los parámetros entre <> deben tomar los valores apropiados para cada objeto.
- Crear un método público boolean equals (Futbolista f) que determine si dos objetos Futbolista representan al mismo jugador (son iguales).
- Debe tener un método público abstracto, jugarConLasManos, que se utilizará para devolver verdadero si el Futbolista puede jugar con las manos y falso en caso contrario (un futbolista sólo puede jugar con las manos si es de la clase Portero).
- Crear los métodos get y set para obtener y dar valor al atributo.

b) Crear la clase Jugador que hereda de Futbolista:

- Debe tener dos atributos públicos, golesMarcados de tipo short y dorsal de tipo byte.
- Crear un constructor que dé valor a todos los atributos, incluidos los heredados y que use el de la clase padre.
- Crear un constructor por defecto, que use el de la clase padre para los atributos heredados, y que ponga como golesMarcados 289 y como dorsal 7.
- Implementar el respectivo método que impone implementar la interface comparable el cual debe retornar la diferencia de edad entre el futbolista y otro futbolista (éste último se pasa como parámetro del método).
- Añadir el código necesario a la clase para que cuando en una clase usuaria se haga System.out.println(elJugador) siendo elJugador un objeto de la clase Jugador, se imprima por pantalla lo siguiente: "El futbolista "+<nombre>+" tiene "+<edad>+ ", y juega de "+<posición>+" con el dorsal "+<dorsal>+ ". Ha marcado "+<goles>. Donde los parámetros entre <> deben tomar los valores apropiados para cada objeto.

c) Crear la clase Portero que hereda de Futbolista:

- Debe tener dos atributos públicos, golesRecibidos de tipo short y dorsal de tipo byte.
- La posición deberá ser siempre "Portero" y podrá jugar con las manos
- Crear un constructor que dé valor a todos los atributos sin asignar, incluidos los heredados y que use el de la clase padre.
- Añadir el código necesario a la clase para que cuando en una clase usuaria se haga System.out.println(elJugador) siendo elJugador un objeto de la clase Portero, se imprima por pantalla lo siguiente: "El futbolista "+<nombre>+" tiene "+<edad>+ ", y juega de

“+<posición>+” con el dorsal “+<dorsal>+ “. Le han marcado “+<golesRecibidos>. Donde los parámetros entre <> deben tomar los valores apropiados para cada objeto.

• Implementar el respectivo método que impone implementar la interface comparable el cual debe retornar la diferencia de golesRecibidos entre el Portero y otro Portero (éste último se pasa como parámetro del método).

Nota 1: Dorsal es el número de la camiseta del jugador.
Nota 2: Todas las clases anteriores se crearan dentro de un paquete llamado futbol