Laboratorio de Software
Práctica nº 2

Temas
- Interfaces
- Polimorfismo

1.- Declaración e implementación de **Interfaces**.

a) ¿Son correctas las siguientes declaraciones?

```
interface ColPrimarios {
 int ROJO=1, VERDE=2, AZUL=4;
}

interface ColArcoIris extends ColPrimarios {
 int AMARILLO=3, NARANJA=5, INDIGO=6, VIOLETA=7;
}

interface ColImpresion extends ColPrimarios {
 int AMARILLO=8, CYAN=16, MAGENTA=32;
}

interface TodosLosColores extends ColImpresion, ColArcoIris {
 int FUCSIA=17, BORDO=ROJO+90;
}

class MisColores implements ColImpresion, ColArcoIris {
   public MisColores() {
      int unColor=AMARILLO;
   }
}
```

b) Analice el código de la interface y las clases que la implementan. Determine si son legales o no. En caso de ser necesario, realice las correcciones que correspondan.

```
public interface InstrumentoMusical {
  void hacerSonar();
  String queEs();
  void afinar(){}
 }

class abstract InstrumentoDeViento implements InstrumentoMusical {
  void hacerSonar(){
     System.out.println("Sonar Vientos");
  }
  public String queEs() {
     return "Instrumento de Viento";
  }
}

class InstrumentoDeCuerda implements InstrumentoMusical {
  void hacerSonar(){
     System.out.println("Sonar Cuerdas");
  }
  public String queEs() {
     return "Instrumento de Cuerda";
  }
}
```

2.- Redefina la clase **PaintTest** *del ejercicio 6 de la práctica 1* de manera de imprimir las figuras geométricas ordenadas de acuerdo al valor de su área. Defina la comparación entre figuras geométricas usando la siguiente regla: una figura **A** es menor una figura **B** si el área de **A** es menor que el área de **B**. Use para ordenar el arreglo de figuras los métodos de ordenación disponibles en la clase **java.util.Arrays**.

3.- Se desea implementar un tipo especial de **HashSet** con la característica de poder consultar la cantidad total de elementos que se agregaron al mismo. Analice y pruebe el siguiente código de manera de corroborar si realiza lo pedido.

```
public class HashSetAgregados<E> extends HashSet<E> {

        private int cantidadAgregados = 0;

     public HashSetAgregados() {
     }

     public HashSetAgregados(int initCap, float loadFactor) {
        super(initCap, loadFactor);
     }

     @Override public boolean add(E e) {
       cantidadAgregados++;
       return super.add(e);
     }

     @Override public boolean addAll(Collection<? extends E> c) {
        cantidadAgregados += c.size();
        return super.addAll(c);
     }
     public int getCantidadAgregados() {
        return cantidadAgregados;
     }
}
```

a) ¿Por qué al agregar al **HashSetAgregados** los elementos de otra colección (mediante el método *addAll*) el método **getCantidadAgregados** retorna el doble?
b) Diseñe e implemente una alternativa para **HashSetAgregados** sin utilizar herencia (utilizando composición). ¿Qué ventajas proporcionaría esta nueva implementación respecto de la original?
c) Se desea implementar otro tipo especial de **Set** con la característica de poder consultar la cantidad total de elementos que se removieron del mismo. Diseñe e implemente una solución que permita fácilmente definir nuevos tipos de **Set** con distintas características.