Ejercicio 5.1 (0.5 puntos)

Supongamos la siguiente interfaz, que especifica que los objetos de las clases que la implementan son funciones reales de una variable real.

```
public interface Funcion {
  public double evaluar(double v);
}
```

1. Crear clases para representar las funciones constante, identidad, suma, multiplicación, potencial, logaritmo, exponencial, y crear otra clase que represente la composición de dos funciones. Todas estas clases deberán implementar la interfaz `Funcion`.

2. Implementar el siguiente método para aproximar uno de los ceros de una determinada función `f` pasada como parámetro.

```
public static double cero(Funcion f, double inicio, double fin,
                          int numIts)
```

   Para ello se debe utilizar el método de la bisección[1]. Puede suponerse que la función toma valores de distinto signo en `inicio` y `fin`. El parámetro `numIts` especifica el número máximo de iteraciones a realizar.

3. Utilizar la función anterior para aproximar un cero de la función $x^2 \log x - x$ en el intervalo [1,2].

Ejercicio 5.2 (0.5 puntos)

Vamos a crear una jerarquía de clases para representar figuras geométricas utilizando un enfoque distinto al visto en clase. Consideremos la siguiente clase abstracta `Figura`.

```
public abstract class Figura {
  protected Punto posicion;
  protected int alto, ancho;
  ...
  public  abstract boolean contienePunto(Punto posicion);
}
```

Los atributos `posicion`, `alto` y `ancho` definen un rectángulo en el plano que delimita la figura geométrica. Además de los métodos de acceso correspondientes, existe un método abstracto `contienePunto` que comprueba si un determinado punto del plano está contenido dentro de la Figura.

1. Crea subclases de `Figura` para representar rectángulos y círculos en el plano.

2. Añade un método `dibujar` a la clase `Figura` para representar la figura geométrica

1 Más información en http://en.wikipedia.org/wiki/Bisection_method

en una ventana. Para ello deberá comprobar qué puntos del rectángulo delimitador están contenidos dentro de la figura, y utilizar el siguiente método contenido en la clase Ventana,

```
public void  dibujarMapaDeBits(double x, double y, boolean[][] dibujo)
```

que representa la matriz de booleanos en una determinada de posición de la ventana. Cada elemento de la matriz de booleanos representa un punto (píxel) de la pantalla. El valor `true` indica presencia de píxel, y el valor `false` indica ausencia de píxel.

3. Implementar las clases `Union` e `Interseccion` para representar uniones e intersecciones de figuras en el plano. Ambas clases deberán heredar de `Figura`.

Ejercicio 5.3 (0.5 puntos)

El objetivo de este ejercicio es diseñar un procedimiento para comparar algoritmos de ordenación. Un algoritmo de ordenación puede caracterizarse por la siguiente interfaz:

```
public interface AlgoritmoOrdenacion {
  public String nombre();
  public void ordenar(double[] array);
}
```

El método `ordenar` debe modificar el array pasado como parámetro para colocar sus elementos en orden ascendente, mientras que el método `nombre` devuelve el nombre del algoritmo implementado en `ordenar`.

1. Implementar tres algoritmos de ordenación distintos. Pueden tomarse tres de la siguiente lista, o considerar otros algoritmos conocidos:

   - *Bubblesort:* http://es.wikipedia.org/wiki/Ordenamiento_de_burbuja
   - Ordenación por selección: http://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n
   - Ordenación por inserción: http://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n
   - *Mergesort:* http://es.wikipedia.org/wiki/Ordenamiento_por_mezcla
   - *Quicksort:* http://es.wikipedia.org/wiki/Quicksort

2. Escribir un método `comparaAlgoritmos`, que reciba un array de objetos que implementen `AlgoritmoOrdenacion` y un array de valores de tipo `double` para ordenar. El método deberá ejecutar cada uno de los algoritmos de ordenación pasados como parámetro sobre este último array. El método ha de medir el tiempo de ejecución de cada algoritmo, e imprimirá los nombres de los algoritmos ejecutados, junto con sus respectivos tiempos de ejecución. Para medir los tiempos de ejecución, se puede utilizar el método `currentTimeMillis()` de la siguiente forma:

```
long inicio = System.currentTimeMillis();
… aplicar algoritmo de ordenación …
long fin = System.currentTimeMillis();
long tiempoEjecucion = fin - inicio;
```