Un punto en el plano se puede representar mediante un par de coordenadas x e y, ambas tomando valores en el conjunto de los números reales. En Java podemos representar un punto en el plano mediante una instancia de la siguiente clase:

```
public class Punto {
    private double x;
    private double y;

    /* métodos */
    /* A completar en el resto de apartados */
}
```

En Java, el constructor de la clase sirve para inicializar los valores de los atributos a la hora de instanciarse un objeto de la misma en tiempo de ejecución.

Programa un constructor para la clase **Punto**. Debe recibir como parámetros las dos coordenadas del punto.

El método **toString**() tiene un significado especial en los objetos Java. Es el método que se utiliza para obtener una representación como cadena de texto de dicho objeto.

Programa el método toString() para que devuelva una cadena de texto con la representación del punto con el siguiente formato: (x, y), donde x e y deben ser reemplazados por sus respectivos valores. El prototipo de dicha función se especifica a continuación:

```
public String toString() {
    /* ... */
}
```

Pista

Recuerda que el operador "+", aplicado a cadenas de texto, permite concatenarlas. Si se concatenan números a cadenas, Java los convierte automáticamente a cadenas de texto para realizar la concatenación.

Crea un paquete llamado **paqPunto** para guardar en él la clase Punto y la clase Prueba que se te pide más adelante.

Programa una clase llamada **Prueba** que tenga un método main para probar el código anterior. Este método ***debe recibir como argumentos*** de línea de comandos las coordenadas x e y, crear un nuevo objeto Punto con dichas coordenadas e imprimir en su salida estándar (la pantalla en este caso) la representación textual de dicho objeto.

El programa debe comprobar que el número de argumentos de línea de comandos recibido sea el correcto.

Pista

El método parseDouble de la clase Double transforma una cadena de texto a un tipo primitivo double.

Es habitual que los atributos de un objeto se declaren como privados (private) para evitar accesos incontrolados desde otros puntos del código. Si es necesario que desde el exterior se acceda a estos atributos, se proporcionan métodos cuyo nombre empieza, por convenio, con las cadenas "get" (acceso en lectura) y "set" (acceso en escritura).

Programa en la clase Punto los siguientes métodos, que devuelven el valor de las coordenadas x e y. El prototipo de dichos métodos se muestra a continuación:

```
public double getX() {
    /* ... */
}
public double getY() {
    /* ... */
}
public void setX(double x) {
    /* ... */
}
public void setY(double y) {
    /* ... */
}
```

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

Cálculo de distancias.

En este apartado vamos a implementar dos métodos auxiliares que nos permitirán calcular distancias entre puntos.

Programa un método de la clase Punto que devuelva la distancia del punto al origen de coordenadas. El prototipo del método se muestra a continuación:

```
public double distanciaAlOrigen() {
/*  completar */
}
```

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

Programa un método en Punto que devuelva la distancia entre el punto representado por la instancia actual del objeto y otra instancia de Punto que se recibe como parámetro. El prototipo del método se muestra a continuación:

```
public double calcularDistancia(Punto otroPunto) {
    /* completar */
}
```

La **fórmula de la distancia entre dos puntos** A(x1, y1) y B(x2, y2) es:

$$d(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

Cálculo de cuadrante.

Programa un método de la clase Punto que devuelva el cuadrante en el que se encuentra el punto.

Devuelve 0 si está en el origen de coordenadas o sobre alguno de los ejes.

Devuelve 1 si está en el primer cuadrante (x e y positivos).

Devuelve 2 si está en el segundo cuadrante (x negativo e y positivo).

Devuelve 3 si está en el tercer cuadrante (x e y negativos).

Devuelve 4 si está en el cuarto cuadrante (x positivo e y negativo).

El prototipo del método se muestra a continuación:

```
public int calcularCuadrante() {
  /* completar */
}
```

Modifica el código de tu clase de prueba para comprobar que su comportamiento es correcto.

Cálculo del punto más cercano.

En este apartado tienes que basarte en los métodos de los apartados anteriores.

Programa un método en Punto que reciba como parámetro un array de objetos de la clase Punto y devuelva una referencia al objeto de dicho array que esté más cercano al punto actual. El prototipo del método se muestra a continuación:

```
public Punto calcularMasCercano(Punto[] otrosPuntos) {
  /* completar */
}
```

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

La clase Triangulo.

Un triángulo está plenamente definido por sus tres vértices. Estos vértices se pueden representar como objetos de la clase Punto.

Declaración de la clase.

En este apartado tienes que basarte en los métodos de los apartados anteriores.

Programa la clase Triangulo con sus atributos (los 3 puntos) y un constructor que reciba como parámetro tres puntos del plano, así como sus getter, setter y toString. Guárdala en un paquete que llamarás **paqTriangulo**, en él guardarás también la clase **Prueba** donde probarás el método que se describe a continuación.

Longitud de los lados.

Programa el método calcularLongitudLados() de la clase Triangulo, que debe devolver un array de tres posiciones, cada una de las cuales debe ser la longitud de uno de los lados del triángulo. El prototipo del método se muestra a continuación:

```
public double[] calcularLongitudLados() {
    /* completar... */
}
```

Crea un ArrayList con algunos triángulos que pedirás al usuario para guardarlos en el programa e imprímelos. (no hace falta menú, solo es para repasar conceptos vistos hasta ahora)