Agregación

Salvador López Mendoza

Octubre de 2022

Conceptos generales

Programación orientada a objetos:

Solución de problemas en base a un conjunto de objetos que interactúan entre ellos.

Conceptos generales:

- Objeto.
- Clase.
- Mensaje.

Filosofía de trabajo: Para resolver un problema se necesita encontrar los objetos que participan en la solución.

No se necesita conocer el código de las clases de objetos utilizadas.

¿Qué hacer cuando no dispongo de los objetos necesarios?

Problema

Se necesita un programa para ayudar a los alumnos de nivel superior a comprender conceptos básicos de geometría analítica.

Los conceptos con los que se trabajará son *punto, línea y triángulo*.

Detalles de los objetos

Un punto. Se define como una *pareja ordenada de números*.
Un punto puede crearse, desplazarse, calcular su distancia con respecto a otro punto, determinar si está alineado con respecto a otros dos puntos, determinar si es igual a otro punto, e imprimirse como pareja ordenada de puntos separados por comas y entre paréntesis.

Una línea recta. Se requieren *dos puntos* cualquiera.
Lo que se desea hacer con ella es crearla, encontrar su ecuación, calcular su pendiente, su ordenada al origen, dadas dos rectas determinar si son paralelas, si son la misma, si son perpendiculares, su punto de intersección y dada una recta y un punto determinar si éste está en la recta.

Un triángulo. Se define como *tres puntos* no alineados.
Se requiere determinar su perímetro, área y tipo (equilátero, escaleno e isósceles).

Metodología de diseño (1)

Encontrar los objetos principales.

Sustantivos: programa, alumno, concepto, geometría, *punto*, *línea*, *triángulo*.

Programa no se considera clase, alumno es sinónimo de usuario y concepto es una palabra que engloba a punto, línea y triángulo.

De la descripción también se puede definir la estructura de los objetos principales. Por ejemplo, un punto está definido como una pareja ordenada de números.

Metodología de diseño (2-I)

Determinar el comportamiento deseado para los objetos principales.

punto:

- Crear. Desplazar.
- Calcular distancia con respecto a otro punto.
- Determinar si está alineado con otros dos puntos.
- Determinar si es igual a otro punto.
- Imprimirlo en formato `(x,y)`.

Metodología de diseño (2-II)

Determinar el comportamiento deseado para los objetos principales.

línea:

Crear. Encontrar su ecuación.
Calcular su pendiente. Calcular su ordenada al origen.
Determinar si un punto pertenece a la recta.
Determinar si dos rectas son paralelas.
Determinar si dos rectas son la misma.
Determinar si dos rectas son perpendiculares.
Calcular el punto de intersección de dos rectas.

Metodología de diseño (2-III)

Determinar el comportamiento deseado para los objetos principales.

triángulo:

- Crear.
- Calcular su perímetro.
- Calcular su área.
- Determinar si es equilátero.
- Determinar si es isósceles .
- Determinar si es escaleno.
- Determinar si es igual a otro.

De la definición del problema se tiene que la estructura del punto son dos números. La estructura de la línea dos puntos y la estructura del triángulo son tres puntos.

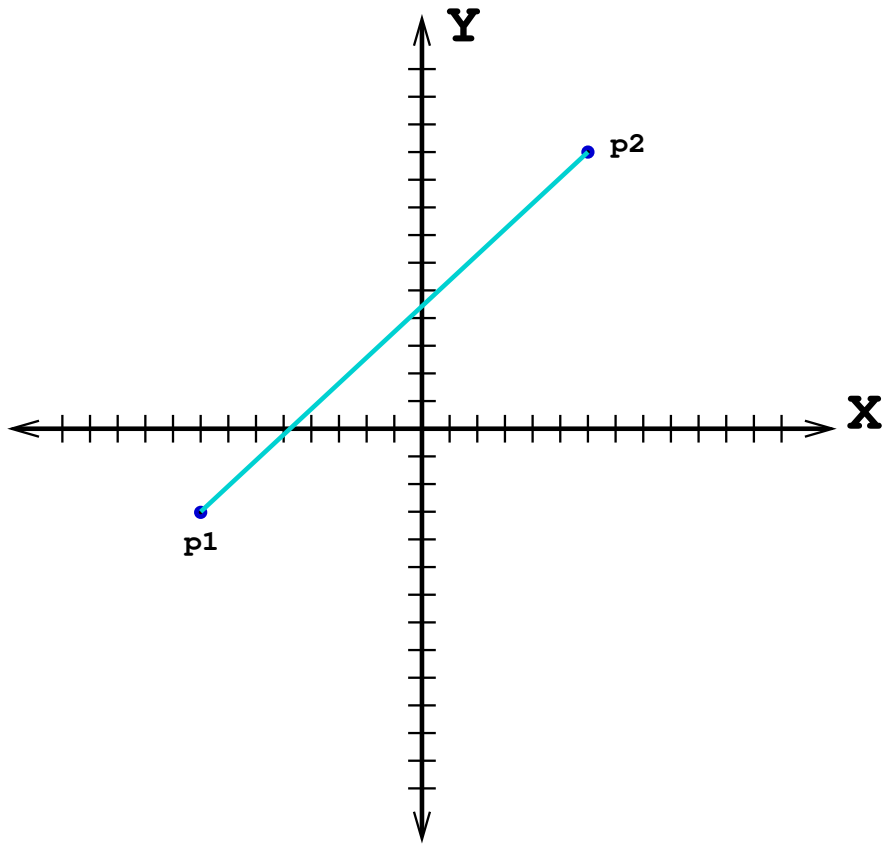
Buscar objetos

¿Qué hacer si no encontramos las clases de los objetos necesarios para resolver mi problema?

¡Definir nuestras clases!

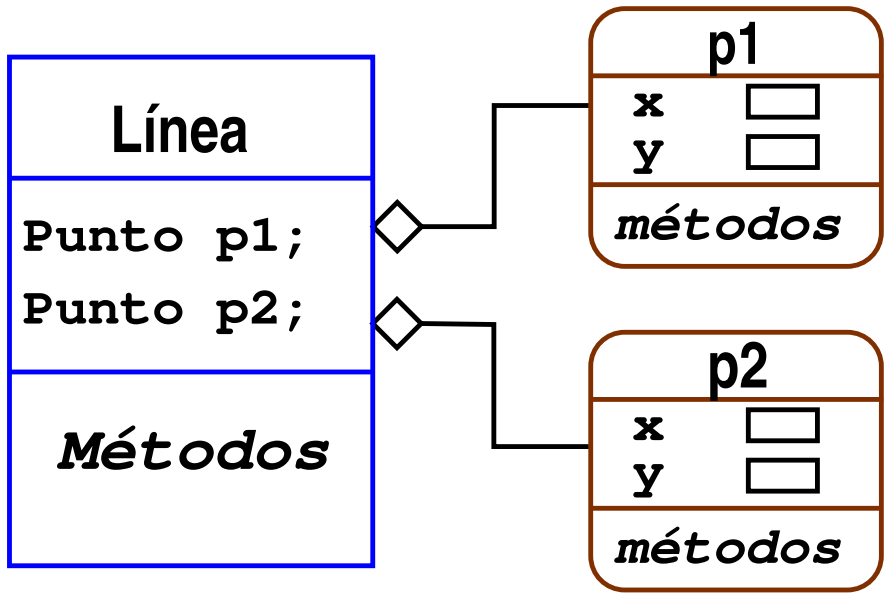
La clase Línea

De la definición del problema se tiene que la estructura de toda línea tendrá un par de puntos como se ilustra en la figura.



La clase Línea (II)

La estructura de toda línea tendrá un par de puntos. La figura ilustra el diagrama de la clase.



Cada atributo es una referencia a un objeto de la clase `Punto`.

La **relación de agregación** también se conoce como una relación *tiene-un*.

Estructura de la clase Línea (en Java)

```java
/**
 * Clase para crear líneas rectas en el plano cartesiano
 * Objetivo: ilustrar relación de agregación entre objetos
 * @author  Amparo López Gaona
 * @see Punto
 */
public class Linea {
  private Punto p1;          //Estructura
  private Punto p2;
                             //Métodos
  ...
}
```

Constructores (1)

Constructor tomando como base dos puntos:

```
/**
 * Constructor de una línea a partir de dos puntos
 * @param p1Ini -- punto de origen
 * @param p2Ini -- segundo punto
 */
public Linea (Punto p1Ini, Punto p2Ini) {
  p1 = new Punto(p1Ini);
  p2 = new Punto(p2Ini);
}
```

Constructores (2)

Constructor sin parámetros:

```
/**
 * Constructor por omisión. Crea la línea que va del
 * punto (0,0) al punto (1,1).
 */
public Linea () {
  this(new Punto(0,0), new Punto(1,1));
}
```

Hay que crear los puntos que definen al objeto de la clase línea.

Constructores (3)

Constructor de copia:

```
/**
 * Constructor de copia
 * @param recta -- Línea que se toma para crear una nueva
 */
public Linea (Linea recta) {
  this(recta.p1, recta.p2);
}
```

Métodos calculadores

¿Cómo calcular la pendiente de una recta que pasa por los puntos (x_1, y_1), (x_2, y_2)?

$$m = (y_2 - y_1)/(x_2 - x_1)$$

Restricciones:

$$x_2 \neq x_1$$

Cuando $x_1 = x_2$ se tiene una recta paralela al eje Y, cuya pendiente es **infinito**.

Cálculo de la pendiente

```
/**
 * Devuelve la pendiente de una recta
 * @return double - La pendiente de la recta
 */
public double pendiente() {
  double divisor = p2.obtenerX() - p1.obtenerX() ;

  if (divisor != 0) {
    return (p2.obtenerY() - p1.obtenerY())/divisor ;
  } else {                              //Recta paralela al eje Y
    return Double.POSITIVE_INFINITY ;
  }
}
```

Ecuación de una recta

Ecuación de una recta: $y = mx + b$

m es la pendiente y b es la **ordenada al origen** (punto en el que la línea corta al eje Y).

```
/**
 * Determina la ecuación de la recta
 * @return String -- La ecuación de la recta
 */
public String ecuacion() {
 if (p1.obtenerX() == p2.obtenerX()) {
   return "x = " + p1.obtenerX();  //Recta paralela al eje Y
 }
 return (ordenada() >= 0) ?
              "y = " + pendiente() + "x + " + ordenada()
            : "y = " + pendiente() + "x "   + ordenada();
}
```

Método calculador ordenada

Calcula la ordenada al origen.

Es el lugar en donde la recta cruza al eje Y.

Valor de y cuando $x = 0$.

```
/**
 * Calcula la ordenada al origen de la recta dada
 * @return double La ordenada al origen
 */
public double ordenada() {
  return (p1.obtenerX() == p2.obtenerX()) ?
              Double.POSITIVE_INFINITY
            : p1.obtenerY() - this.pendiente()*p1.obtenerX();
}
```

Determinar si un punto es parte de la recta

Dos casos:

1. Recta paralela al eje Y.
   Se verifica que la abscisa sea igual a la de cualquiera de los puntos p_1 y p_2.
2. Se sustituyen las coordenadas del punto que se recibe como parámetro en la ecuación de la recta.
   Si se cumple la igualdad el punto es parte de la recta.

¿Y si se usan los métodos de la clase punto?

Determinar si un punto es parte de la recta (II)

```java
/**
 * Determina si un punto pertenece a la recta
 * @param p - Punto a determinar si está en la recta
 * @return boolean - true si el punto está en la recta
 *                   y false en otro caso
 */
public boolean contiene(Punto p) {
  return p.estanAlineados(p1,p2) ;
}
```

Método para determinar si dos rectas son iguales

Se verifica que las pendientes son iguales.

Las rectas deben tener un punto en común: **la ordenada al origen.**

```
/**
 * Determina si dos líneas son la misma
 * @param linea1 -- Linea con la que se comparará
 *                  la línea original
 * @return boolean -- true si son la misma línea,
 *                    false en otro caso
 */
public boolean equals(Linea linea1) {
  return pendiente() == linea1.pendiente() &&
         this.ordenada() == linea1.ordenada() ;
}
```

Método para determinar si dos líneas son paralelas

Dos líneas son paralelas si tienen la misma pendiente.

```
/**
 * Determina si dos líneas son paralelas
 * @param linea1 -- Linea con la que se comparará la
 *                  línea original
 * @return boolean -- true si son paralelas y
 *                    false en otro caso.
 */
public boolean esParalelaA(Linea linea1) {
  return this.pendiente() == linea1.pendiente() ;
}
```

Método para determinar si dos líneas son perpendiculares

La pendiente de una es el negativo del inverso multiplicativo de la otra.

$$m = -1/m_1$$

$$m \times m_1 = -1$$

En caso de que una pendiente sea igual a infinito, la otra debe ser igual a cero.

$$m = \infty \Rightarrow m_1 = 0$$

Método para determinar si dos líneas son perpendiculares

```
/**
 * Determina si dos líneas son perpendiculares
 * @param linea1 -- Linea con la que se compara
 * @return -- true si son perpendiculare,
 *            false en otro caso
 */
public boolean esPerpendicularA(Linea linea1) {
  double m = this.pendiente() ;
  double m1 = linea1.pendiente() ;
  final double INFINITO = Double.POSITIVE_INFINITY ;
  boolean perpendicular ;

  perpendicular = ( (m == 0 && m1 == INFINITO) ||
                    (m1 == 0 && m == INFINITO) ) ;
  if (!perpendicular) { perpendicular = ( m*m1 == -1 ) ; }
  return perpendicular ;
}
```

El punto de intersección entre dos rectas

Si las líneas son paralelas, el punto de intersección es ∞.

La primera línea tiene la ecuación:

$$y = m_1 x + b_1$$

La segunda línea tiene la ecuación:

$$y = m_2 x + b_2$$

Se determinan los puntos despejando: $m_1 x + b_1 = m_2 x + b_2$

Despues se evalúa la ecuación de la recta (cualquiera de las dos) para el valor de x que se ha encontrado.

El punto de intersección entre dos rectas (II)

```
/**
 * @param linea1 - la segunda línea
 * @return Punto- punto de intersección
 */
public Punto interseccion(Linea linea1) {
  double nuevaX, nuevaY ;
  if (this.esParalelaA(linea1)) {
    nuevaX = nuevaY = Double.POSITIVE_INFINITY ;
  } else {
    nuevaX = (this.ordenada() - linea1.ordenada()) /
             (linea1.pendiente() - this.pendiente()) ;
    nuevaY = this.pendiente()*nuevaX + this.ordenada() ;
  }
  return new Punto(nuevaX, nuevaY) ;
}
```