Creación de clases

Salvador López Mendoza

Septiembre 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.

Metodología de diseño (3). Definir escenarios

Escenario: Inicio del programa

1. El programa da la bienvenida al usuario y le muestra un menú con los diferentes temas que puede trabajar.
2. El programa solicita al usuario el tema sobre el que se desea trabajar.
3. El usuario indica el tema deseado.
4. El programa envía un mensaje para trabajar con el tema indicado.

Metodología de diseño (3). Definir escenarios

Escenario: Trabajar con puntos.

1. El programa solicita al usuario las coordenadas de un punto.
2. El usuario proporciona las coordenadas de un punto.
3. El programa crea un punto.
4. El programa muestra al usuario un menú con las diferentes operaciones que puede hacer con un punto.
5. Dependiendo del método elegido por el usuario, el programa solicita más datos.
6. El programa muestra el resultado de la operación elegida.

Buscar objetos

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

¡Definir nuestras clases!

Definición de clases

En una clase se define la **estructura** y el **comportamiento** que tendrán sus objetos.

Estructura Conjunto de elementos que caracterizan a los objetos de cada clase.
Aquí se guarda el **estado** de cada objeto.

Comportamiento Acciones que pueden realizar los objetos de la clase.

Definición de clases (II)

Definición de clases. Estructura

Conjunto de elementos que caracterizan a los objetos de cada clase.

- ¿Cuántas cosas caracterizan a los objetos de esta clase?
- ¿Cómo es cada una de esas características?

El **proceso de abstracción** ayuda a determinar cuáles son las características relevantes.

Se utilizan **variables** de los tipos primitivos para representar las características de los objetos.

El **estado del objeto** está determinado por los valores de cada una de las características que lo definen.

Definición de clases. Comportamiento

Acciones que pueden realizar los objetos de la clase.

Cada acción que pueda realizar un objeto de la clase requiere de un **método**.

Los métodos pueden entregar un *resultado*.

En muchas ocasiones se requiere de *parámetros* que proporcionan información al método. Esa información les permite realizar su trabajo.

Definición de clases. Comportamiento (II)

Los métodos se agrupan de acuerdo a su función:

Constructores. Sirven para crear objetos de la clase.

Observadores. Permiten conocer el estado del objeto.

Modificadores. Permiten alterar el estado del objeto.

Especiales. Métodos comunes a todas las clases.

Calculadores. Implementan el comportamiento particular de los objetos de la clase.

Definición de clases en Java

Las clases en Java constan de:

Encabezado. Nombre de la clase.

Cuerpo. Atributos y métodos.

Encabezado de una clase

Elementos del encabezado:

Visibilidad

Pública (`public`).
Privada (`package`).

Palabra reservada `class`

Nombre de la clase

Sustantivo en singular, iniciando con máyuscula.

Cuerpo de la clase. Atributos

Para definir la estructura de los objetos de una clase se declara un conjunto de atributos.

- Variables de los tipos básicos del lenguaje.
- Referencias a objetos de otras clases.

A las variables usadas para definir los atributos de una clase se les llama ***variables del ejemplar***.

La estructura de los objetos de la clase no cambia.

El estado de cada objeto puede cambiar a lo largo de su existencia.

Características de los atributos

Al usuario del objeto sólo le interesa su *comportamiento* y su *identidad*.

La estructura es privada.

Se puede acceder al *estado del objeto* a través de los métodos de la clase.

La **encapsulación** consiste en organizar los elementos de un objeto (atributos y métodos) de manera que algunos sean accesibles desde el exterior y otros privados.

La ventaja de la encapsulación es que el usuario de la clase desconoce la representación del objeto y si ésta cambia, en principio, no le afecta.

Componentes de los atributos

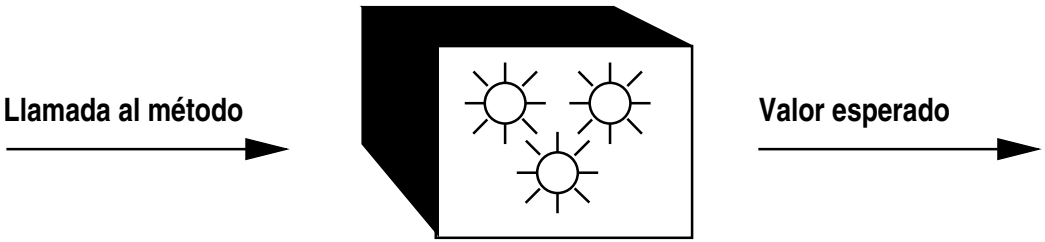
- Visibilidad.
  Privado (`private`). Sólo se puede acceder a él desde los métodos que son parte de la clase.
  Público (`public`). Atributo visible desde fuera de la clase.
  Si el atributo es una constante, puede ser público.
  Protegido (`protected`).
- Calificador `static`. Es un atributo compartido por todos los ejemplares de la clase. Es opcional.
- Calificador `final`. Es un atributo constante. Es opcional.
- Tipo. Nombre de algún tipo primitivo o nombre de una clase.
- Nombre. Identificador del atributo. Debe empezar con letra minúscula.
- Valor inicial. Es opcional.

Cuerpo de la clase. Métodos

El comportamiento de los objetos de una clase se programa mediante la creación de métodos.

Un **método** es el conjunto de instrucciones (programación del algoritmo) que se deben realizar al llamar a ejecutar tal método, es decir, al enviar el mensaje correspondiente.

Un método es visto por el objeto cliente como una caja negra con nombre y una función específica.



Encabezado del método

- Visibilidad. Especifica si se puede tener acceso al método desde un objeto cliente.
  `public, private` o `protected`.
- Calificador `static`. Indica si el método es del objeto o de la clase. Es opcional.
- Tipo del valor que devuelve. El tipo es el nombre de una clase, un tipo primitivo o bien `void`, en caso de no devolver nada.
- Nombre. Es un identificador como cualquier otro, de preferencia un nombre que describa la función del método y por convención empieza con una letra minúscula.
- Elementos necesarios para realizar esta tarea. Estos elementos se especifican como una lista y se llaman **parámetros**.

El nombre de un método y su lista de parámetros formales se conoce como la **firma del método**.

Parámetros

Los parámetros sirven para que un método pueda realizar su trabajo tomando como base un conjunto de valores distintos en cada ocasión.

Al definir un método se usa una lista de parámetros que sirven para indicarle *cuántos son* y de *qué tipo*.

Cada parámetro tiene un **nombre**.

Dentro del método se puede hacer referencia a cada parámetro por su nombre.

A los parámetros que se usan para la definición de método se les llama **parámetros formales**.

A los parámetros que se usan al invocar al método se les llama **parámetros actuales**.

Paso de parámetros

Al invocar al método se deben especificar tantos parámetros como se usaron en la definición del mismo.

Cuando hay una cantidad incorrecta de parámetros, el compilador indica que hay un error.

La correspondencia entre los parámetros formales y los parámetros actuales es de acuerdo a la posición que ocupan.

Los parámetros actuales deben ser del mismo tipo de los parámetros formales (en la correspondencia uno a uno).

Al llamar a ejecución un método, el valor del parámetro actual se asigna como valor inicial del parámetro formal y termina la relación entre ambos parámetros.

Paso de parámetros (II)

En Java se usa **paso de parámetros por valor**.

> *Al invocar a un método, se evalúa cada parámetro y se pasa el valor resultante.*

Si en el cuerpo del método se modifica el valor del parámetro formal no cambia el valor del parámetro actual.

Debido a este funcionamiento, el parámetro actual puede ser una variable, una literal o una expresión; en este último caso se evalúa la expresión y su valor se asigna al parámetro formal.

Cuerpo de los métodos

Es un bloque con las instrucciones que implementan la tarea que debe realizar.

El cuerpo del método es privado, en el sentido que no se conoce fuera de la clase que lo implementa.

Es importante que cada método implemente una y sólo una tarea.

Relacionado con el *alcance de una variable* está su **tiempo de vida**.

Una vez que termina la ejecución de un método, el parámetro formal y su valor se pierde.

El tiempo de vida de una variable del ejemplar es el mismo que el del objeto al que pertenece.

Cuerpo de los métodos (II)

Elementos que se pueden usar dentro de un método:

- Cualquier atributo de la estructura del objeto de la clase.
- Parámetros y variables locales.
  Tanto la variable como su valor se pierden una vez que termina la ejecución del bloque en que se declaró.
  Los parámetros formales están disponibles sólo dentro del cuerpo del método en que se declaran.
  Los parámetros son como variables locales, sólo que éstos cuentan con un valor inicial dado al momento de llamar al método.
  El alcance de las variables del ejemplar es toda la clase en la que se definen, por eso pueden utilizarse en cualquier método de la clase.
- Datos obtenidos como respuesta a mensajes enviados a objetos de su misma clase o de otras clases.
- Atributos definidos como públicos o estáticos en otras clases.

Clasificación de los métodos

- Métodos constructores.
- Métodos observadores (de acceso).
- Métodos modificadores.
- Métodos calculadores.

Todos los métodos deben estar bien comentados.

Empezando con un comentario como preámbulo del método (siguiendo los lineamientos de `javadoc`).

Métodos observadores (de acceso)

Métodos que se usan para conocer las características de los objetos (los valores de los atributos).

No reciben parámetros.

Estos métodos devuelven un valor. Es del mismo tipo que el atributo a que corresponden.

Se recomienda que su nombre empiece on la palabra en inglés `get` o con la palabra `obtener`; esto va seguido del (posible) nombre del atributo.

Métodos modificadores

Se recomienda escribir un método modificador por cada atributo que se defina dentro de la clase.

En su forma más sencilla, cada método modificador recibe como parámetro el nuevo valor del atributo para el que se ha definido el método.

El parámetro debe ser del mismo tipo que el atributo.

Estos métodos no devuelven valor.

Mantienen la integridad de los objetos de la clase al ser la única forma en que un cliente puede modificar el valor de un atributo.

Se recomienda que el nombre del método empiece con la palabra `set` o `asignar`.

Métodos calculadores

Estos métodos se emplean para implementar cualquier comportamiento deseado de los objetos.

Pueden recibir y/o devolver valores y trabajan con el estado del objeto para calcular un valor.

Al escribir un método se debe tener presente que los métodos se llaman con un objeto, sobre cuyo estado se está trabajando.

Métodos constructores

Un **constructor** es un método cuyo objetivo es asignar valor inicial a cada atributo de un objeto recién creado.

Se garantiza que el objeto se crea con un estado válido.

Un constructor se distingue de cualquier otro método porque:

- Tiene el mismo nombre que la clase a la que pertenece.
- No tiene especificado valor de regreso, ni siquiera `void`.
- La única forma de llamarlo es usando el operador `new`.

Es común que las clases tengan más de un constructor, con lo cual se permiten diferentes posibilidades de creación de un objeto.

Cuando tenemos varios métodos con el mismo nombre pero distinta lista de parámetros se dice que estamos usando **sobrecarga de métodos**.

Métodos especiales

Algunos métodos siempre deben existir en las clases que desarrollemos.

- Para probar si una clase funciona correctamente:

  El método `main`.
- Para comparar objetos de una clase:

  El método `equals`.
- Para imprimir el objeto:

  El método `toString`.

El método *main*

El método `main` es indispensable, empieza la ejecución del programa.

Lo común es que en este método esté expresado el algoritmo para resolver el problema original.

Sintaxis del encabezado del método `main`:

```
public static void main(String [] pps) {
....  //Algoritmo
}
```

El método *main* (II)

Un método estático, como `main`, no requiere un objeto para ser llamado, pues es un método de la clase, no de los objetos.

Como no opera en el contexto de un objeto particular no puede referenciar variables del ejemplar.

Lo que sí puede hacer el método `main` es referenciar variables estáticas.

En el `main` se puede acceder sólo a variables estáticas o variables locales.

Puede invocar la ejecución de métodos estáticos o métodos aplicados a objetos ya creados.

El método *equals*

Permite comparar si el estado de dos objetos es el mismo.

No es suficiente comparar las referencias a los objetos. El resultado es verdadero si se hace referencia al mismo objeto.

Lo que se necesita en la mayoría de las ocasiones es saber si los valores de todos los atributos son iguales, aunque sean objetos distintos.

Compara uno a uno los valores de los atributos.

Firma del método `equals`:

```
public boolean equals(Object)
```

El parámetro es de la clase `Object` porque se puede comparar un objeto de una clase con otro objeto de cualquier otra clase.

El método *toString*

En la mayoría de las ocasiones necesitamos conocer (visualmente) el estado completo de un objeto.

La forma más sencilla de hacerlo es imprimirlo.

Posibilidad 1: *imprimir cada uno de los atributos*.

```
Punto p = new Punto(3.25, -1.5) ;
System.out.print("Valor del atributo 1:" + p.obtenerX()) ;
```

Posibilidad 2: *imprimir todo el objeto*.

```
Punto p = new Punto(3.25, -1.5) ;
System.out.println( p ) ;
```

El método `println` recibe como parámetro una cadena de caracteres.

El método *toString* (II)

Para todo objeto se necesita obtener una representación como cadena de caracteres.

Toda clase debe tener un método llamado `toString`.

Firma del método `toString`:

```
public String toString()
```

Este método se invoca automáticamente al colocar la referencia al objeto en el lugar en el que se espera una cadena de caracteres.

Probando las clases

¿Cómo saber si una clase funciona correctamente o no?

Los programadores confiamos en nuestro trabajo.

Existen metodologías de desarrollo de programas en las que se trata de escribir programas que son correctos desde su diseño.

Se pueden cometer distintos tipos de errores:

- Errores al escribir.
- Errores al traducir el algoritmo al lenguaje de programación.
- Errores al desarrollar el algoritmo.

Los dos primeros tipos de error (sintaxis) los indica el compilador.

¿Cómo detectar un error?

Probando el funcionamiento de los métodos

Se establece un conjunto de pruebas para los distintos métodos.

Los casos de prueba se basan en la descripción del problema.

Generalmente se usan casos en los que se conoce de antemano el resultado.

Se sugiere probar con casos tipo (los más frecuentes).

También se puede probar con casos extremos.

Los que podrían generar una situación de error.