Práctica 3

Arrays de objetos: cálculo del área de un polígono

Marzo de 2017

DECSAI

Complementos de Programación

Curso 2016/2017

Índice

1. Introducción

En esta práctica cubriremos los siguientes objetivos:

- Seguir practicando con la construcción y uso de clases.
- Usar arrays de objetos.
- Aprender a usar la clase `ArrayList` para guardar una lista de objetos.
- Comprender por qué el uso de datos privados ayuda a que una clase sea fácil de modificar en el futuro, sin que los programas que la usan tengan que modificarse.
- Ejecutar un programa java a partir de un fichero `.jar`.
- Ejecutar un programa redirigiendo la entrada estándar desde un fichero.

El programa que vamos a crear permitirá calcular el perímetro y el área de un polígono convexo, con un número de vértices cualquiera. Para cada vértice, usaremos la clase `Punto` que hicimos en la práctica anterior.

- Crear un nuevo proyecto en *netbeans*. Para ello seleccionamos la opción `Menú File -> New Project (Ctrl-Shift-N)`.
- Seleccionar `Java > Java Application` y pinchamos en el botón **Next**.
- Usa `CalculoAreaPoligono` como nombre del proyecto.
- Como carpeta para el proyecto selecciona por ejemplo:

```
/home/usuario/ComplementosProgramacion/ProjectsNB/Practica3
```

- Marca la casilla *Create Main Class*.
- Pulsa el botón **Finish**.

Tras estos pasos, se habrá creado la clase `CalculoAreaPoligono`, con un método `main()` vacío.

Añade una nueva clase al proyecto, la clase `Punto` que nos permitirá representar los vértices del polígono. Añade también la clase `Poligono` para representar objetos polígono.

El proyecto estará ahora formado por tres clases, la clase `Punto`, la clase `Poligono` y la clase `CalculoAreaPoligono`.

Ejercicio 1 *Copia en la clase* `Punto` *los datos miembro y métodos que hiciste para la práctica anterior.*

Un polígono convexo (en que todos sus ángulos miden menos de 180 grados) puede representarse mediante la siguiente clase:

```
public class Poligono{
   private Punto ptos[]; // array con los vértices del Poligono
   private int numPuntos; // número de vértices que tiene ahora el Poligono
   ...
}
```

donde `numPuntos` tiene el número de puntos que tiene actualmente el polígono. Los lados del polígono están constituidos por los pares de puntos (`ptos[i]`, `ptos[i+1]`) ($i = 0, \ldots, numPuntos - 2$) y un par adicional (`ptos[0]`, `ptos[numPuntos-1]`).

Ejercicio 2 *Implementa los siguientes métodos en la clase* `Poligono`*:*

- *Constructor sin parámetros para crear un polígono vacío. El constructor creará el array de puntos con una capacidad inicial para* `4` *vértices. También inicializará a* `0` *el dato miembro* `numPuntos` *para indicar que aun no contiene ningún punto. Este constructor no debe crear los vértices (puntos) del polígono. Un usuario de esta clase podrá añadir los vértices con el método* `addPunto` *descrito a continuación.*

- `public void addPunto(Punto p)` *que añade un punto (vértice) a un* `Poligono`*. Antes de añadir el punto, el método debe comprobar si es posible añadirlo (o sea, si cabe en el array). Si no cabe, deberá crearse un nuevo array para los puntos con el doble de capacidad a la que tenga actualmente, y se copiarán los puntos antiguos en el nuevo array.*

  El punto pasado como parámetro se añadirá en la primera posición disponible del array (en posición `numPuntos`*).*

- `public int numberOfPuntos()` *que devuelve el número de vértices que tiene actualmente el Poligono.*

- `public Punto getPunto(int n)` *que devuelve el n-ésimo punto de un Poligono. Devuelve* `null` *si ese punto no existe en el polígono.*

- `public String toString()` *que devuelva un* `String` *con el número de vértices y las coordenadas* `x` *e* `y` *de cada vértice. Entre cada dos números, pondremos un separador (un espacio en blanco o el caracter* `\n`*). Por ejemplo, a continuación mostramos el* `String` *de un polígono que contiene 6 vértices:*

```
6
4 12
10 12
14 6
10 0
4 0
0 6
```

Ejercicio 3 *Ahora modifica la función* `main()` *de la siguiente forma. Solicitará por la entrada estándar el número de puntos* n *de un polígono, a continuación creará un objeto* `Poligono` *vacío y pedirá que se introduzcan las coordenadas de los* n *puntos por la entrada estándar, añadiéndolos uno a uno. Usaremos el método* `leer(Scanner conin)` *de la clase* `Punto` *para leer cada punto. El programa mostrará en la salida estándar, el polígono introducido, haciendo uso del método* `toString()` *de la clase* `Poligono`*. Compila y ejecuta el programa para comprobar el funcionamiento.*

2. Interfaz básico e interfaz adicional de una clase

Uno de los objetivos que debemos seguir cuando se diseña una clase, es que sea fácil de mantener (de modificar en el futuro). El hacer `private` sus datos miembro ayuda a conseguirlo. En nuestro caso, la clase `Poligono` ha seguido esta recomendación. De esta forma, solo los métodos de la clase `Poligono` pueden usar estos datos miembro (*ocultamiento de información*).

También puede ser interesante limitar el número de métodos de instancia de una clase que usan directamente la parte privada de esa clase. Aquellos métodos que puedan construirse sin acceder directamente a la parte privada de la clase, sería recomendable que no lo hicieran. Estos métodos se construirían solo haciendo uso de otros métodos de la clase. A veces, por razones de eficiencia, este tipo de métodos también usan directamente los datos miembro privados de la clase.

Podemos decir que los métodos que necesitan obligatoriamente acceder a los datos miembro constituyen el *interfaz básico* de la clase y los métodos que pueden construirse sin hacerlo constituyen el *interfaz adicional*.

Ejercicio 4 *Haz un método para calcular el perímetro de un* `Poligono`*. Este método se implementará sin usar directamente los datos miembro privados de la clase* `Poligono`*.*

- `public double perimetro()`

Modifica el `main()` *para que calcule e imprima el perímetro del polígono introducido. Compila y ejecuta el programa para comprobar el funcionamiento.*

3. Redirección de la entrada estándar y ficheros jar

Cuando ejecutamos un programa desde una shell de un sistema operativo, es posible redirigir la entrada estándar para que la tome de un fichero en lugar del teclado. Esto permite que tengamos los datos de entrada

a un programa guardados en un fichero de texto, y que lo podamos ejecutar sin necesidad de introducirlos por teclado. La forma de redirigir la entrada estándar desde un fichero `ficheroConDatos.txt` es añadirlo al comando que ejecuta el programa para indicar que queremos tomar la entrada de ese fichero.

```
java Programa < ficheroConDatos.txt
```

Ejercicio 5 *Ejecuta el programa del cálculo del perímetro de un polígono redirigiendo la entrada estándar a partir del fichero* `hexagono.txt` *que puedes descargar de la plataforma docente. Para ello ejecutaremos el programa desde un terminal (shell) usando el fichero* `.jar` *que genera Netbeans en la carpeta* `dist` *de nuestro proyecto, tras compilarlo.*

Un fichero `.jar` *contiene todas los ficheros* `.class` *de nuestro programa, lo que facilita su distribución. Este fichero incluye también un fichero de texto manifest (*`MANIFEST.MF`*) que contiene información sobre cómo se ejecutará el programa. En particular contiene una línea que indica el nombre de la clase principal (la que contiene el método* `main()` *que queremos ejecutar).*

```
Main-Class: calculoareapoligono.CalculoAreaPoligono
```

En este caso, `calculoareapoligono` *es el paquete donde está incluida la clase principal que se llama* `CalculoAreaPoligono`*.*

Para ejecutar el programa usando el fichero `.jar`*, redirigiendo la entrada estándar de* `hexagono.txt` *usaremos:*

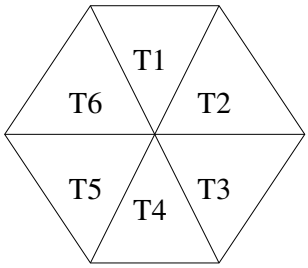
```
java -jar CalculoAreaPoligono.jar < hexagono.txt
```

El fichero `hexagono.txt` *contiene en primer lugar el número de puntos de un polígono regular de seis lados, y luego las coordenadas* `x` *e* `y` *de los vértices separadas por espacios en blanco:*

```
6
4 12
10 12
14 6
10 0
4 0
0 6
```

4. Cálculo del área de un polígono

El área de un polígono puede calcularse mediante la suma de las áreas de un conjunto de triángulos que están inscritos en él. Los triángulos se obtienen uniendo los lados del polígono con un punto interior, mediante dos segmentos, tal como muestra la siguiente figura.



Ejercicio 6 *Añadir un método que calcule el área de un polígono a través de la suma de las áreas de sus triángulos constituyentes:*

- `public double area()`

Este método se implementará también sin usar directamente los datos miembro privados de la clase `Poligono`*.*

Para implementarlo, primero hay que construir un método auxiliar ***static*** *que devuelva el área del triángulo formado por los puntos pto1, pto2 y pto3:*

- `private static double areaTriangulo(Punto pto1, Punto pto2, Punto pto3)`

Se usará la siguiente fórmula para calcular el área:

$$area = \sqrt{T(T-S_1)(T-S_2)(T-S_3)}$$

$$T = \frac{S_1 + S_2 + S_3}{2}$$

donde S_1*,* S_2 *y* S_3 *son las longitudes de los lados del triángulo*

Debe implementarse también el método auxiliar (sin usar directamente los datos miembro privados de la clase `Poligono`*):*

- `private Punto getPuntoInterior()`

que devuelve un punto interior cualquiera del polígono. Este método puede implementarse como el punto medio de la recta entre dos vértices opuestos o bien simplemente devolviendo uno de los vértices del polígono, ya que un vértice es también un punto interior al `Poligono`*.*

Ejercicio 7 *Modifica el método* `main()` *para que se calcule e imprima también el área del polígono introducido. Compila y ejecuta el programa para comprobar el funcionamiento.*

El método `main()` *debe haber quedado con el siguiente aspecto:*

```
public static void main(String[] args){
  Poligono  poligono;
  Scanner conin = new Scanner(System.in);
  int nPuntos = leer número puntos de la entrada estándar

  poligono = new Poligono();

  // bucle para leer los puntos (repetir nPuntos veces) de entrada estándar
     // leer un punto de la entrada estándar
     // añadirlo a  poligono

  // calcular e imprimir perímetro de  poligono

  // calcular e imprimir área de  poligono

}
```

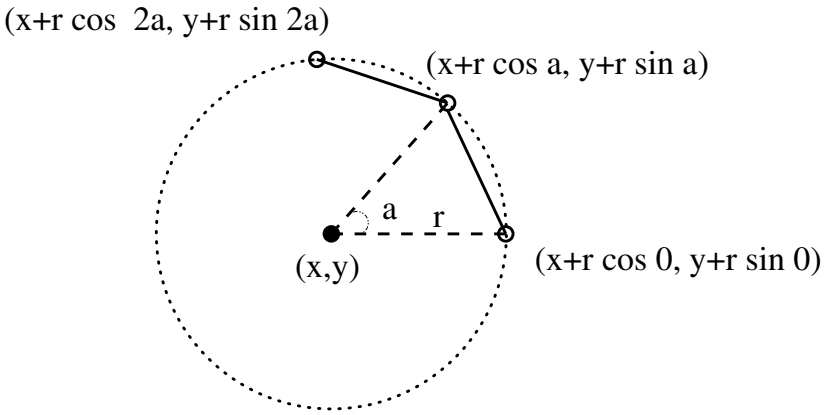
5. Aproximación del área de un círculo mediante el área de un polígono

En esta sección haremos un programa para aproximar el área de un círculo mediante el área de un polígono regular inscrito en el círculo (ver figura de más adelante).

Ejercicio 8 *Añade una nueva clase al proyecto, la clase* `Circulo`*. Copia los datos miembro, el constructor y métodos que hiciste para esta clase en la práctica anterior.*

Ejercicio 9 *Desarrolla un constructor para la clase* `Poligono` *que permita construir un polígono regular de* **n** *lados, a partir de un objeto* `Circulo`*. En java, podemos obtener el seno o coseno de un ángulo* a *expresado en radianes, mediante el uso de los métodos estáticos de la clase* ***Math****:*

- `public static double sin(double a)`
- `public static double cos(double a)`



Ejercicio 10 *Crea una nueva clase* `CalculoAreaCirculo`*. Esta clase contendrá únicamente un método* `main()` *con el código necesario para calcular de forma aproximada el área de un círculo mediante el área de un polígono. Para ello, solicitará al usuario, el radio de un círculo y las coordenadas* `x` *e* `y` *de su centro, para crear un objeto de la clase* `Circulo`*. Puedes usar para ello el método* `leer()` *de la clase* `Circulo`*.*

También solicitará el número de lados **n** *que queremos que tenga el polígono. El programa debe escribir ahora el área del polígono, y el área del círculo.*

El método `main()` *debe haber quedado con el siguiente aspecto:*

```
public static void main(String[] args){
  Circulo circulo;
  Poligono  poligono;
  Scanner conin = new Scanner(System.in);

  circulo = new Circulo();
  // leer radio y coordenadas x e y del círculo
  // leer numero lados n del polígono usado para aproximar el área del círculo

  poligono = new Poligono(circulo, n); // Poligono para aproximar el círculo
```

```
  // Mostrar área de anterior Poligono (área aproximada del círculo)

  // Mostrar el área del Circulo con método area() de clase Circulo
}
```

Compila y ejecuta el programa para comprobar el funcionamiento.
Puedes comprobar el funcionamiento del programa redirigiendo la entrada estándar a partir del fichero `circulo.txt` *disponible en la plataforma docente.*

```
3 7 6
12
```

6. Cambio de la representación interna de una clase: clase ArrayList

6.1. Clase ArrayList

La clase `ArrayList` es una clase cuyos objetos pueden usarse para almacenar una lista de objetos al igual que los arrays. Pero a diferencia de los arrays, el tamaño no necesita fijarse al crearlo.

Mostramos a continuación algunos de los métodos de esta clase. Para más detalle consultar la web https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html:

- `ArrayList()`: Crea una lista vacía.
- `void add(E o)`: Añade un nuevo elemento `o` al final de esta lista.
- `void add(int index, E o)`: Añade un nuevo elemento `o` en la posición especificada por `index` en esta lista.
- `void clear()`: Borra todos los elementos de esta lista.
- `boolean contains(Object o)`: Devuelve `true` si esta lista contiene el elemento `o`.
- `E get(int index)`: Devuelve el elemento de esta lista en la posición `index`.
- `int indexOf(Object o)`: Devuelve la posición de la primera ocurrencia de `o` en esta lista.
- `boolean isEmpty()`: Devuelve `true` si esta lista no contiene ningún elemento.
- `int lastIndex(Object o)`: Devuelve el índice de la última ocurrencia de `o` en esta lista.
- `boolean remove(Object o)`: Borra el primer elemento `o` de esta lista. Devuelve `true` si el elemento es borrado.

- `int size()`: Devuelve el número de elementos en esta lista
- `boolean remove(int index)`: Borra el elemento en la posición especificada. Devuelve `true` si se borra el elemento.
- `E set(int index, E o)`: Pone el elemento en la posición `index`.

Se dice que la clase `ArrayList` es una clase genérica con tipo genérico `E`. Al crear un `ArrayList`, podemos especificar cualquier tipo concreto para reemplazar a `E`. Por ejemplo, la siguiente sentencia crea un `ArrayList` para almacenar objetos `String`:

```
ArrayList<String> nombresAlumnos =
                          new ArrayList<String>();
```

Desde la versión JDK 7, la anterior sentencia puede simplicarse sin incluir el tipo concreto en el constructor, debido a la característica conocida como *inferencia de tipo*:

```
ArrayList<String> nombresAlumnos = new ArrayList<>();
```

El siguiente código muestra un ejemplo de uso de un `ArrayList` para almacenar `String`s.

```
import java.util.ArrayList;

public class TestArrayList {
  public static void main(String[] args) {
    // Crea una lista para almacenar ciudades
    ArrayList<String> listaCiudades = new ArrayList<String>();

    // Añadir algunas ciudades a la lista
    listaCiudades.add("Madrid");
    // listaCiudades ahora contiene [Madrid]
    listaCiudades.add("Granada");
    // listaCiudades ahora contiene [Madrid, Granada]
    listaCiudades.add("Ubeda");
    // listaCiudades ahora contiene [Madrid, Granada, Ubeda]
    listaCiudades.add("Almeria");
    // listaCiudades ahora contiene [Madrid, Granada, Ubeda, Almería]
    listaCiudades.add("Motril");
    // contiene [Madrid, Granada, Ubeda, Almeria, Motril]
    listaCiudades.add("Linares");
    // contiene [Madrid, Granada, Ubeda, Almería, Motril, Linares]

    System.out.println("Tamanio de la lista? " + listaCiudades.size());
    System.out.println("Esta Almería en la lista? " +
      listaCiudades.contains("Almería"));
    System.out.println("La posición de Granada en la lista? "
      + listaCiudades.indexOf("Granada"));
    System.out.println("Esta la lista vacía? " +
      listaCiudades.isEmpty()); // Imprime false

    // Inserta una nueva ciudad en la posición 2
    listaCiudades.add(2, "Armilla");
    // contiene [Madrid, Granada, Armilla, Ubeda, Almería, Motril, Linares]

    // Borra una ciudad de la lista
    listaCiudades.remove("Almería");
    // contiene [Madrid, Granada, Armilla, Ubeda, Motril, Linares]

    // Borra una ciudad en la posición 1
    listaCiudades.remove(1);
    // contiene [Madrid, Armilla, Ubeda, Motril, Linares]
```

```
    // Muestra el contenido de la lista
    System.out.println(listaCiudades.toString());

    // Muestra el contenido en la lista en orden inverso
    for (int i = listaCiudades.size() - 1; i >= 0; i--)
      System.out.print(listaCiudades.get(i) + " ");
    System.out.println();
  }
}
```

La salida de este programa es la siguiente:

```
Esta Almería en la lista? true
La posición de Granada en la lista? 1
Esta la lista vacía? false
[Madrid, Armilla, Ubeda, Motril, Linares]
Linares Motril Ubeda Armilla Madrid
```

La clase `ArrayList` permite que un programa sea más fácil de implementar que con arrays por dos razones:

- El tamaño de un `ArrayList` es flexible, por lo que no es necesario especificar su tamaño por adelantado, como en un array.
- La clase `ArrayList` contiene muchos métodos útiles. Por ejemplo, podemos comprobar si una lista contiene un elemento usando el método `contains`. En un array, deberíamos nosotros escribir el código para ello.

Podemos recorrer todos los elementos de un `ArrayList` usando un bucle *foreach*:

```
for (tipoElemento elemento: arrayList){
  // Procesar el elemento
}
```

6.2. Cambio de la representación interna de una clase

En el siguiente ejercicio demostraremos que es sencillo cambiar la representación interna de una clase si sus datos miembro los habíamos declarado como `private`. Crearemos una nueva clase llamada `Poligono2` que será una modificación de la clase `Poligono`. Tendrá los mismos métodos y constructores, pero cambiarán los datos miembro privados (representación interna de la clase). Crearemos también una nueva clase `CalculoAreaPoligono2` cuyo código será una copia de la clase `CalculoAreaPoligono` pero usará `Poligono2` en lugar de `Poligono`. Haremos lo mismo con la clase `CalculoAreaCirculo2` (copia de la clase `CalculoAreaCirculo` pero que usará `Poligono2` en lugar de `Poligono`).

Con esto queremos demostrar que los programas que usaban la clase `Poligono` siguen funcionando de la misma forma tras cambiar su representación interna, sin necesidad de modificarlos.

Ejercicio 11 *Crea una nueva clase* `Poligono2` *copiándola de la clase* `Poligono`*. Modifica la representación para que en lugar de un array, se use un* `ArrayList` *para almacenar los puntos del polígono. Será necesario modificar también los siguientes métodos:*

- *Constructor sin parámetros*
- *Constructor* `Poligono(Circulo c, int n)`
- `public void addPunto(Punto p)`
- `public int numberOfPuntos()`
- `public Punto getPunto(int n)`
- `public String toString()`

Crea dos nuevas clases `CalculoAreaPoligono2` *y* `CalculoAreaCirculo2` *cuyos códigos serán copias exactas de las clases* `CalculoAreaPoligono` *y* `CalculoAreaCirculo2` *pero usando* `Poligono2` *en lugar de* `Poligono`*.*

Comprueba el funcionamiento de estos dos nuevos programas. Para ejecutarlos desde la línea de comandos podemos usar el fichero `.jar` *de la carpeta* `dist` *de nuestro proyecto Netbeans de la siguiente forma:*

```
java -cp CalculoAreaPoligono.jar
         calculoareapoligono.CalculoAreaPoligono2

java -cp CalculoAreaPoligono.jar
         calculoareapoligono.CalculoAreaCirculo2
```

Hemos usado la opción `-cp` *para indicar el* `CLASSPATH`*, lugar donde el intérprete de java debe buscar las clases del programa a ejecutar.* `CalculoAreaPoligono2` *y* `CalculoAreaCirculo2` *son los programas a ejecutar en este caso.*

También podríamos redirigir la entrada estándar en los anteriores comandos usando:

```
java -cp CalculoAreaPoligono.jar
         calculoareapoligono.CalculoAreaPoligono2
         < hexagono.txt

java -cp CalculoAreaPoligono.jar
         calculoareapoligono.CalculoAreaCirculo2
         < circulo.txt
```