IIP (E.T.S. de Ingeniería Informática)
Curso 2018-2019

Práctica 7. Arrays: una aplicación de gestión de un grupo de polígonos en el plano

Profesores de IIP
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Objetivos y trabajo previo a la sesión de prácticas

Esta práctica resume todos los conceptos vistos en la asignatura y, en particular, los correspondientes al *"Tema 7. Arrays: definición y aplicaciones"*.

Su realización presupone que, con anterioridad al trabajo del laboratorio, se ha leído detenidamente la descripción del problema y la organización de clases que se propone para su resolución.

La práctica dura tres sesiones en las que se desarrollarán las actividades propuestas en este boletín. Durante las dos primeras se debería poder completar el código de las clases `Polygon` y `PolygonGroup` planteadas, dejando la tercera sesión para ultimar el código que hubiera quedado pendiente de las sesiones anteriores, completar el programa `Test7` y realizar las últimas pruebas y correcciones.

2. Descripción del problema

En esta práctica se va a abordar el problema de gestionar un grupo de polígonos, que se suponen dispuestos sobre un plano.

Cada polígono viene dado por la secuencia de sus vértices (puntos en el plano), y es de un determinado color de relleno (color de la superficie del polígono) de forma que, a partir de dichos datos, se puede dibujar en un espacio de dibujo como los de la librería gráfica `Graph2D` utilizada en las prácticas anteriores. Un polígono se podrá trasladar en el plano y cambiarle el color de relleno.

Un grupo de polígonos es, como su propio nombre indica, un agrupamiento de polígonos al que se podrán añadir nuevos elementos, borrar elementos existentes, o tratar de forma solidaria a todos los elementos del grupo. También se puede seleccionar un polígono del grupo para tratarlo individualmente, por ejemplo, para desplazarlo o cambiarle el color.

Un comportamiento habitual en las aplicaciones gráficas en las que se maneja un grupo de figuras, polígonos en el caso que nos ocupa, es que se van superponiendo por el orden en que se añaden al grupo. Así pues, de haber solapamiento entre las superficies de los polígonos, abajo del todo se encuentra el más antiguo y arriba del todo el más reciente. Por ejemplo, el dibujo de la figura 1 representa gráficamente un grupo de polígonos en el que se han añadido, por orden, un rectángulo verde, un triángulo azul, un rectángulo rojo y un cuadrilátero amarillo.

Este orden se puede cambiar a conveniencia, seleccionando un polígono para enviarlo al fondo, traerlo delante del todo, etc.

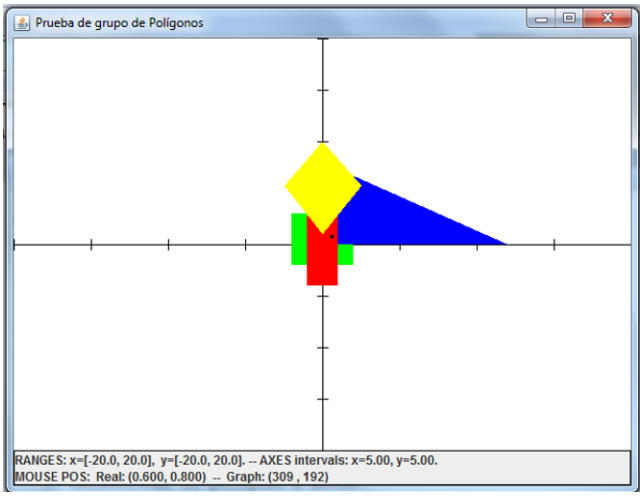

Figura 1: Grupo de polígonos.

Cuando se ha representado gráficamente un grupo de polígonos, la manera en que se puede seleccionar un polígono del grupo es clicando sobre un punto de su superficie que esté a la vista, es decir, que no quede oculto por otros polígonos superpuestos. En la parte gráfica de esta práctica, se detectará el punto del clic de manera que, en el grupo de figuras, conocido dicho punto, se podrá acceder al polígono señalado. Para ello, el grupo de polígonos revisará sus elementos por orden inverso, de más arriba a más abajo, buscando el primero que contenga a dicho punto: este es el polígono a seleccionar, dado que se superpone al resto de polígonos que contuvieran el mismo punto. En el ejemplo de la figura 1 se ve cómo, para señalar el rectángulo rojo, se ha clicado uno de los puntos visibles de su superficie, en concreto el de coordenadas `(0.6,0.8)`.

Para dar cuenta del comportamiento descrito, en esta práctica se va a desarrollar una pequeña aplicación formada por las clases que se muestran en la figura 2, y que son:

- Clase `Polygon`. Representación y tratamiento de polígonos en el plano.
- Clase `PolygonGroup`. Representación y tratamiento de un grupo de polígonos.
- Clase `Test7`. Programa de prueba que crea un grupo de figuras y permite probar diferentes acciones sobre el grupo. Para ayudar a comprobar el efecto de dichas acciones, este programa visualiza gráficamente los resultados, usando la librería `Graph2D` del paquete `graph2D`.

Actividad 1: preparación del paquete BlueJ `pract7`

1. Descargar los ficheros `Polygon.java`, `GroupPolygon.java`, `Test7.java` disponibles en la carpeta de material para la práctica 7 de *PoliformaT*.

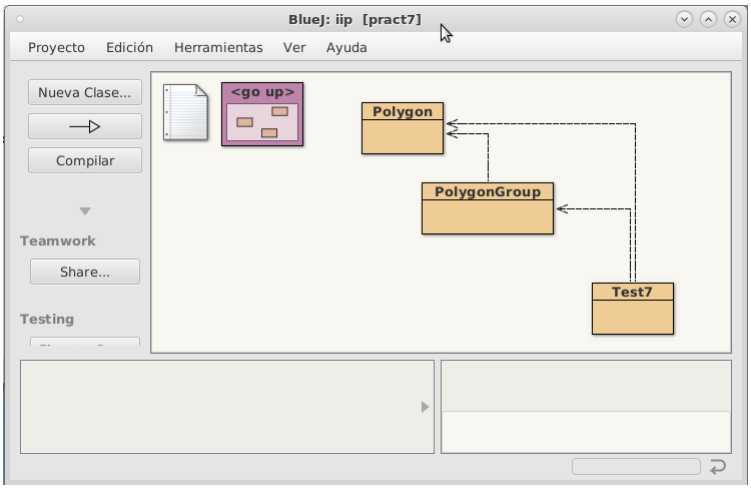

Figura 2: Clases del paquete `iip[pract7]`.

2. Abrir el proyecto *BlueJ* de trabajo de la asignatura (`iip`) y crear un nuevo paquete `pract7`.

3. Agregar al paquete `pract7` las clases descargadas con la opción (*Edición - Agregar Clase desde Archivo*). Comprobar que sus primeras líneas incluyen la directiva `package pract7;`, que indica que son clases del paquete.

3. La clase `Polygon`

Un `Polygon` viene dado por una secuencia de n vértices, $v_0, v_1, \ldots, v_{n-1}$, de forma que sus lados son los segmentos $\overline{v_0v_1}, \overline{v_1v_2}, \ldots, \overline{v_{n-2}v_{n-1}}, \overline{v_{n-1}v_0}$. Además, tiene un color de relleno.

La clase se define mediante los siguientes atributos de instancia privados:

- `v`: un array de objetos de tipo `Point` (la clase desarrollada en la práctica 5), en el que se almacenarán en posiciones sucesivas los n vértices de tipo `Point` del polígono. Para cada polígono de la clase, la longitud de `v` deberá de ser del número n de vértices o lados del polígono.

- `color`: un color de relleno, de tipo `Color` del paquete `java.awt`.

En la figura 3 se muestra un ejemplo de `Polygon` y la representación gráfica del polígono correspondiente.

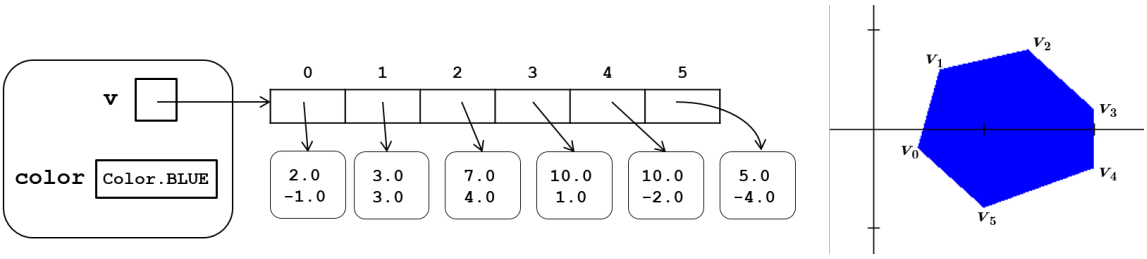

Figura 3: Un `Polygon` y su representación gráfica.

Los métodos de la clase son los que se describen a continuación:

- Constructor `public Polygon(double[] x, double[] y)`: Construye un `Polygon` a partir de un array `x`, con las abscisas $\texttt{x}_0$, $\texttt{x}_1$, $\texttt{x}_2$, ..., $\texttt{x}_{n-1}$ de sus vértices, y un array `y`, con las ordenadas

$\texttt{y}_0$, $\texttt{y}_1$, $\texttt{y}_2$, ..., $\texttt{y}_{n-1}$ de sus vértices, siendo $n > 0$. Los vértices definen un polígono cuyos lados se extienden de un vértice al siguiente, y cerrándose en $(\texttt{x}_0, \texttt{y}_0)$. Por ejemplo, si `x` es el array `{2.0, 3.0, 7.0, 10.0, 10.0, 5.0}` e `y` el array `{-1.0, 3.0, 4.0, 1.0, -2.0, -4.0}`, debe construir el objeto de la figura 3.

Por defecto, el polígono es de color azul (`Color.BLUE`).

- Métodos `public Color getColor()` y `public void setColor(Color nC)`, consultor y modificador del color, respectivamente.
- Métodos `public double[] verticesX()` y `public double[] verticesY()` que, respectivamente, devuelven un array con las sucesivas abscisas de los vértices y un array con las sucesivas ordenadas de los vértices.
- Método `public void translate(double incX, double incY)`, que traslada los vértices del polígono: `incX` en el eje X, `incY` en el eje Y.
- Método `public double perimeter()`, que devuelve el perímetro del polígono.
- Método `public boolean inside(Point p)`, que, aplicando el *algoritmo del rayo* que se discute en el apéndice A, comprueba si el `Point p` es interior al polígono. Si el punto es interior al polígono devuelve `true`, y si el punto es exterior al polígono devuelve `false`.

Actividad 2: implementación y prueba de la clase `Polygon`

Implementar los atributos y métodos de la clase `Polygon`. Notar que, para poder usar las clases `Point` y `Color`, se han incluído las siguientes cláusulas de importación:

```
import java.awt.Color;
import pract5.Point;
```

Los métodos se irán probando a medida que se vaya complentando su código. En primer lugar, se comprobará el método constructor, creando un triángulo de vértices $(0,0)$, $(0,3)$ y $(4,0)$, e inspeccionando el objeto obtenido, como se muestra en la figura 4.

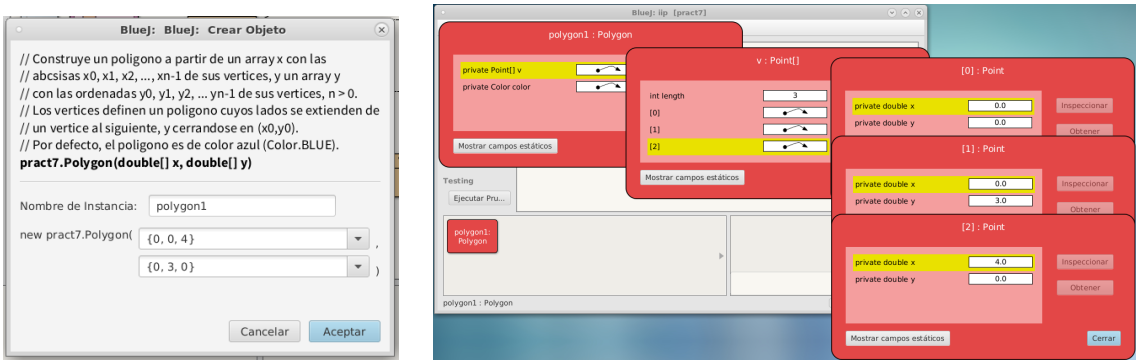

Figura 4: Construcción y examen de un `Polygon` en *BlueJ*.

A continuación, se irán probando los métodos de instancia, sobre todo los que manejan los vértices del polígono, como `verticesX`, `verticesY`, `translate` y `perimeter`, tal como se muestra para este último en la figura 5.

En el caso de `translate`, si se prueba a trasladar el triángulo anterior `1.0` en abscisas y `1.0` en ordenadas, se debe comprobar que `verticesX` y `verticesY` devuelven, respectivamente, los arrays `{1.0, 1.0, 5.0}` y `{1.0, 4.0, 1.0}`.

Para poder probar el método `inside`, en la zona de código se importará la clase `Point` del paquete `pract5`, y se creará un `Point` de coordenadas $(1,1)$, que se podrá arrastrar al banco de objetos para usarlo como parámetro del método (ver figura 6). Como el punto es interior al triángulo,

el resultado debe ser `true`. Repetir las pruebas con los puntos $(4, 5)$ (punto exterior) y $(-1, 3)$ (punto exterior para el que el rayo pasa por el vértice $(0, 3)$ del triángulo).

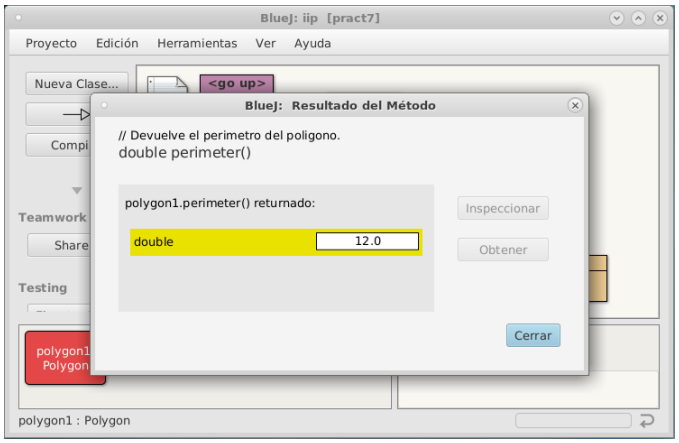

Figura 5: Prueba del método `perimeter` sobre el triángulo de la figura 4.

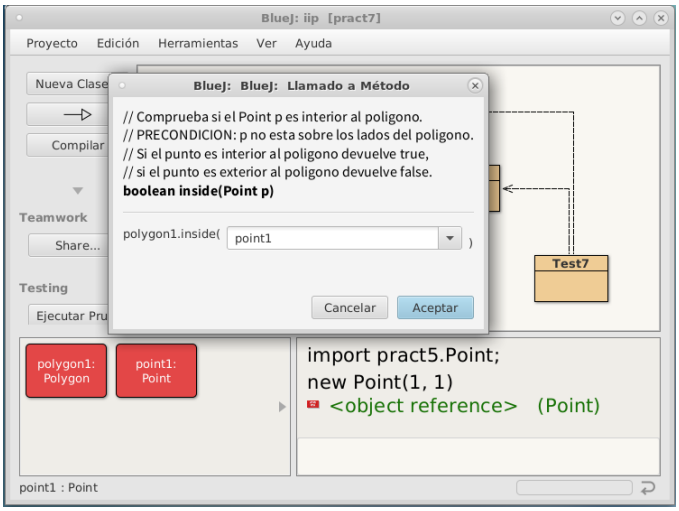

Figura 6: Prueba del método `inside` sobre el triángulo de la figura 4 y el `Point (1.0,1.0)`.

4. La clase `PolygonGroup`

Un `PolygonGroup` vendrá dado por la secuencia de polígonos que forman parte del grupo, que estará ordenada por el orden de inserción en el grupo, de más antiguo a más reciente. Todos los grupos tendrán una capacidad o número máximo de polígonos.

La clase se define mediante los siguientes atributos:

- `MAX`: atributo público de clase constante, de tipo `int`, que da la capacidad del grupo, iniciada a 10.
- `group`: atributo privado de instancia, un array de objetos de tipo `Polygon` y longitud `MAX`, en el que se almacenarán, en posiciones sucesivas y por orden de inserción, los polígonos del grupo. El resto de posiciones del array se mantienen a `null`.
- `size`: atributo privado de instancia, la talla o número de polígonos en el grupo.

En la figura 7 se muestra un objeto `PolygonGroup` que corresponde al grupo de la figura 1.

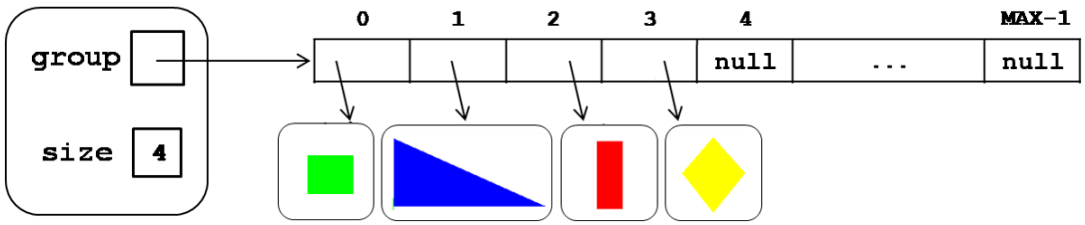

Figura 7: Un `PolygonGroup` de talla 4.

Los métodos de la clase son los que se describen a continuación:

- Constructor `public PolygonGroup()`: Construye un `PolygonGroup` vacío, iniciando el array `group` a un array de longitud `MAX` (con todas sus componentes a `null`, por defecto) y dejando la talla `size` a 0.
- Método `public void add(Polygon pol)`, que añade al grupo, arriba del todo, el polígono `pol`. Si se excede la capacidad del grupo, entonces el polígono no se puede añadir y el método no hace nada.
- Método `public int getSize()`, consultor de la talla del grupo.
- Método auxiliar `private int search(Point p)`, que busca en el grupo descendentemente, de más arriba a más abajo, el primer polígono que contiene a `p`. Si lo encuentra, devuelve su índice en el array `group`; si no existe, devuelve -1.

  Este método se usará en los métodos `remove`, `toFront`, `toBack`, `translate` siguientes, para encontrar la posición en el array `group` del polígono señalado por un punto `p`.
- Método `public void remove(Point p)`, que elimina del grupo el polígono seleccionado mediante el punto `p`.

  Tanto este método como los tres siguientes no harán nada si no hubiera ningún polígono en el grupo que contuviera a `p`.

  Notar que en la implementación de este método se deberá procurar que los polígonos restantes en el grupo después de la eliminación, continúen apareciendo en posiciones consecutivas del array, desde la 0 en adelante, y por orden de inserción. La componente del array siguiente al último polígono restante en el grupo se deberá poner a `null`.
- Método `public void toFront(Point p)`, que sitúa al frente del grupo, arriba del todo, el polígono seleccionado mediante el punto `p`.
- Método `public void toBack(Point p)`, que sitúa al fondo del grupo, abajo del todo, el polígono seleccionado mediante el punto `p`.
- Método `public void translate(Point p, double incX, double incY)`, que traslada en el plano el polígono seleccionado mediante el punto `p`. Las abscisas de sus vértices se incrementan en `incX` y las ordenadas en `incY`.
- Método `public Polygon[] toArray()`, que devuelve un array de longitud igual a la talla del grupo con la secuencia de polígonos del grupo, por orden desde el de más abajo al de más arriba.

Actividad 3: implementación y prueba de la clase `PolygonGroup`

Implementar los atributos y métodos de la clase `PolygonGroup`. Notar que, como en la clase `Polygon`, para poder usar la clase `Point` y `Color`, se han incluído las cláusulas de importación:

```
import java.awt.Color;
import pract5.Point;
```

Al ir acabando los métodos, se irán haciendo las pruebas correspondientes. En primer lugar, se creará e inspeccionará el grupo vacío, tal como se muestra en la figura 8.

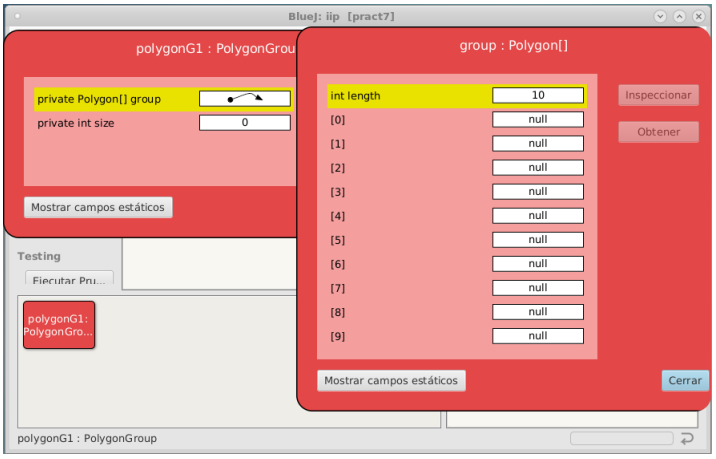

Figura 8: Prueba del constructor de `PolygonGroup`.

Para probar los métodos `add`, `size`, `toArray`, `search` y `remove`, por simplicidad se crearán tres triángulos idénticos de coordenadas $(0,1),(2,2)$ y $(0,3)$ (arrays de abscisas `{0.0, 2.0, 0.0}` y de ordenadas `{1.0, 2.0, 3.0}`) y se añadirán al grupo (tres triángulos perfectamente superpuestos), inspeccionando el grupo a medida que va creciendo. En la figura 9 se muestra el estado que se debe observar tras añadir los tres polígonos.

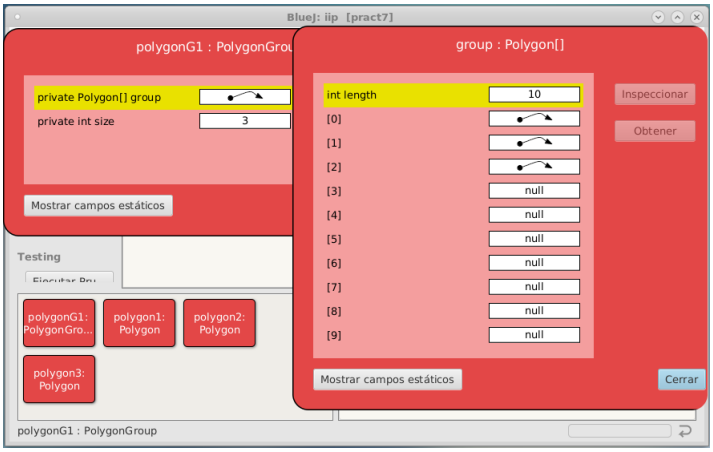

Figura 9: Prueba del método `add` de `PolygonGroup` e inspección del resultado.

Comprobar a continuación que `toArray` devuelve un array de tres polígonos (ver figura 10).

Seguidamente, se deben hacer tres operaciones de eliminación del primer polígono que contenga el punto $(1,2)$, inspeccionando el grupo tras cada eliminación. Dado que los tres polígonos son idénticos y contienen dicho punto, se debe comprobar que se elimina en primer lugar el que ocupa la posición 2 del array `group` (el más reciente), en segundo lugar el que ocupa la posición 1 (el siguiente más reciente) y, finalmente, el que ocupa la posión 0 (el más antiguo), quedando el grupo vacío. Una última operación de eliminación seguiría dejando el grupo vacío.

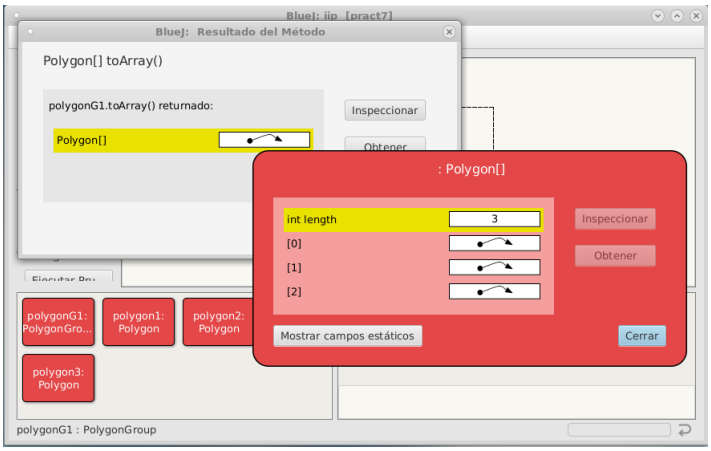

Figura 10: Prueba del método `toArray` de `PolygonGroup`.

En la figura 11 se observa cómo queda el grupo después de hacer la primera de las llamadas a `remove`.

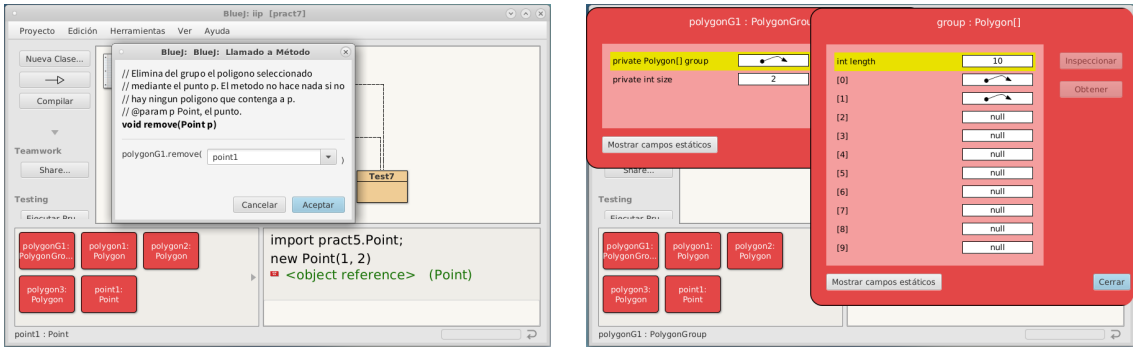

Figura 11: Prueba de búsqueda y eliminación de un polígono: llamada al método `remove` de `PolygonGroup` e inspección del resultado.

Tras estas pruebas preliminares, se podrá hacer una batería de pruebas más completa para los métodos de la clase con el programa `Test7` de la siguiente sección.

5. La clase `Test7`

La clase `Test7` es un programa que, además de probar diferentes acciones sobre un grupo de polígonos, facilita la visualización del comportamiento de dichas acciones mediante una representación gráfica. Para ello, tiene implementado un método `drawGroup` que muestra el grupo en un espacio de dibujo de la librería `Graph2D`. También hace uso del método `nextMousePressed` de `Graph2D` para recuperar las coordenadas del punto clicado en la ventana de dibujo y, de este modo, seleccionar de forma sencilla un polígono del grupo.

En el programa se crea un grupo de polígonos como el de la figura 1, sobre el que se podrán hacer las pruebas.

Actividad 4: finalización y prueba del método `main` de `Test7`

La clase incluye unos métodos `menu(Scanner)` y `submenu(Scanner)` que permiten al usuario seleccionar diferentes acciones a probar sobre el grupo de polígonos. Se debe leer detenidamente el método `main` y completar su código para que se ejecute el método de `PolygonGroup` que corresponda, según el caso seleccionado.

A continuación se ejecutará el programa para poder hacer tantas pruebas como sea necesario, a través de las diferentes opciones del menú.

6. Validación de las clases

Cuando tu profesor lo considere conveniente, dejará disponibles en *PoliformaT* unas clases de prueba para validar tu código. En ese caso, tendrás que descargar los archivos correspondientes sobre el directorio del paquete `pract7` y reabrir tu proyecto `iip` desde *BlueJ*.

En general, para pasar los tests correctamente, hay que asegurarse de que se usan los mismos identificadores de atributos y métodos propuestos en este documento y en la documentación de los archivos `.java` que se te proporcionan, siguiendo estrictamente las características sobre modificadores y parámetros propuestos en la cabecera de los mismos.

Actividad 5: validación de la clase `Polygon`

1. Para usar el test de esta actividad, debes haber validado previamente el método `cross` de la clase `Point` (se debe copiar en el paquete `pract5` el test de validación `PointUnitTest.class`), ya que se utiliza en el método `inside` de la clase. Cualquier error en la clase `Point` probablemente causaría errores en el test de la clase `Polygon`.

2. Elige la opción *Test All* del menú contextual que aparece al hacer clic con el botón derecho del ratón sobre el icono de la clase *Unit Test*. Como siempre, se ejecutarán un conjunto de pruebas sobre los métodos implementados de la clase correspondiente, comparando resultados esperados con los realmente obtenidos.

3. Al igual que en prácticas anteriores, si los métodos están bien, aparecerán marcados con el símbolo ✓ (green color) en la ventana *Test Results* de *BlueJ*. Por el contrario, si alguno de los métodos no funciona correctamente, entonces aparecerá marcado mediante el símbolo `X`. Si seleccionas cualquiera de las líneas marcadas con `X`, en la parte inferior de la ventana se muestra un mensaje más descriptivo sobre la posible causa de error.

4. Si, tras corregir errores y recompilar la clase, el icono de la *Unit Test* está rayada a cuadros, entonces cierra y vuelve a abrir el proyecto *BlueJ*.

Actividad 6: validación de la clase `PolygonGroup`

Para pasar el test de la clase `PolygonGroup`, elige la opción *Test All* del menú contextual que aparece al hacer click con el botón derecho del ratón sobre el icono de la *Unit Test* de la clase correspondiente. Al igual que en la actividad anterior, se ejecutarán un conjunto de pruebas sobre los métodos implementados de la clase `PolygonGroup`, comparando resultados esperados con obtenidos. Para pasar este test, es requisito imprescindible que las clases `Point` y `Polygon` estén correctamente implementadas. Por lo tanto, asegúrate de haber pasado primero los tests de esas dos clases anteriormente.

A. El método `inside`

Como se comentó en la práctica 5, el método del rayo es un algoritmo basado en el teorema de Jordan, que permite de manera sencilla comprobar si un punto es interior o exterior a un polígono:

Se lanza un rayo desde el punto y se cuenta el número de veces que el rayo atraviesa el perímetro del polígono; teniendo en cuenta que los cruces *de entrada* y *de salida* del rayo en el polígono se alternan en la dirección del rayo, y que el último cruce es necesariamente de salida, entonces:

- Si el número de cruces es par, el número de veces que el rayo entra en el polígono es igual al número que veces que sale, por lo que el punto está fuera del polígono.

- Si es impar, hay un cruce más de salida que de entrada, por lo que el punto está dentro.

En la figura 12 se ve un ejemplo de cómo un rayo lanzado desde un punto interior al polígono, cruza su perímetro un número impar de veces.

El método se puede implementar recorriendo todos los lados del polígono y contando el número de lados que son atravesados por el rayo. Sólo hay que tener la precaución de contabilizar correctamente el caso en que el rayo atraviesa un vértice, punto que pertenece a dos lados del polígono. En ese caso, se puede considerar lo siguiente:

- Si, como sucede con el vértice v_8 del polígono de la figura 12, el vértice conecta dos lados, uno por el extremo más bajo y otro por el más alto, entonces el rayo atraviesa el perímetro, y sólo se debe contar un cruce.
- Si, como sucede con los vértices v_3 y v_6 del mismo polígono, el vértice conecta dos lados, ambos por el extremo más bajo, o ambos por el extremo más alto, entonces no atraviesa el perímetro, y no se debe contar ningún cruce o, equivalentemente, contar un número par.
- Notar que si el rayo transcurre por un lado del polígono paralelo al eje X, no se debe contar ningún cruce de dicho lado.

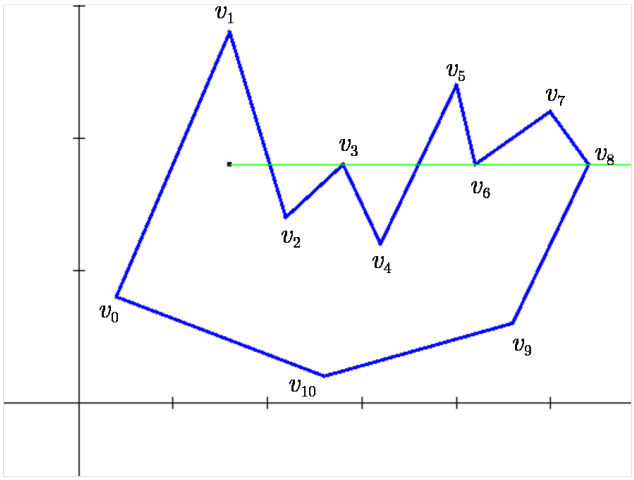

Figura 12: Método del rayo.

Para tener en cuenta esta observación con respecto a los vértices, se puede adoptar el convenio de contar un cruce del perímetro cuando se atraviesa un lado, por ejemplo, por el extremo más bajo y no contarlo cuando se atraviesa un lado por el extremo más alto (o viceversa).

La siguiente iteración realiza el conteo de cruces de los lados de un polígono por un rayo lanzado desde un punto `p`, usando el método `cross` de la clase `Point` y adoptando el convenio anterior para los vértices:

```
int nCuts = 0, n = v.length;
int cross;
for (int i = 0; i < n - 1; i++) {
    cross = p.cross(v[i], v[i + 1]);
    if (cross == Point.CROSS || cross == Point.LOW_CROSS) { nCuts++; }
}
cross = p.cross(v[n - 1], v[0]);
if (cross == Point.CROSS || cross == Point.LOW_CROSS) { nCuts++; }
```

Así pues, la implementación del método `inside` de `Polygon` debe realizar este conteo, y terminar devolviendo `true` si `nCuts` es impar, y `false` en caso contrario.