PRG (E.T.S. de Ingeniería Informática)
Curso 2019-2020

Práctica 5. Secuencias enlazadas: una aplicación de gestión de un grupo de polígonos en el plano

(2 sesiones)

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Objetivos y trabajo previo a la sesión de prácticas

Esta práctica es un ejercicio de desarrollo de una estructura de datos, en la que se usarán las secuencias enlazadas presentadas en el último tema de la asignatura. La implementación de los métodos de la clase desarrollada dará pie a ejercitar los algoritmos básicos sobre tales secuencias: inserción y eliminación de datos, recorridos y búsquedas, y otros cambios sencillos en el orden de los datos.

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 dos sesiones en las que se desarrollarán las actividades propuestas en este boletín. Durante la primera sesión se deberían poder completar las actividades 1 a 5 planteadas, y las actividades 6 y 7 durante la segunda sesión.

2. Descripción del problema

En la práctica 7 de IIP se abordaba el problema de gestionar un grupo de polígonos, que se suponían dispuestos sobre un plano, y que se describía de la siguiente forma.

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`[1] del paquete `graph2D`. 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, como 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 al frente (delante del todo), etc.

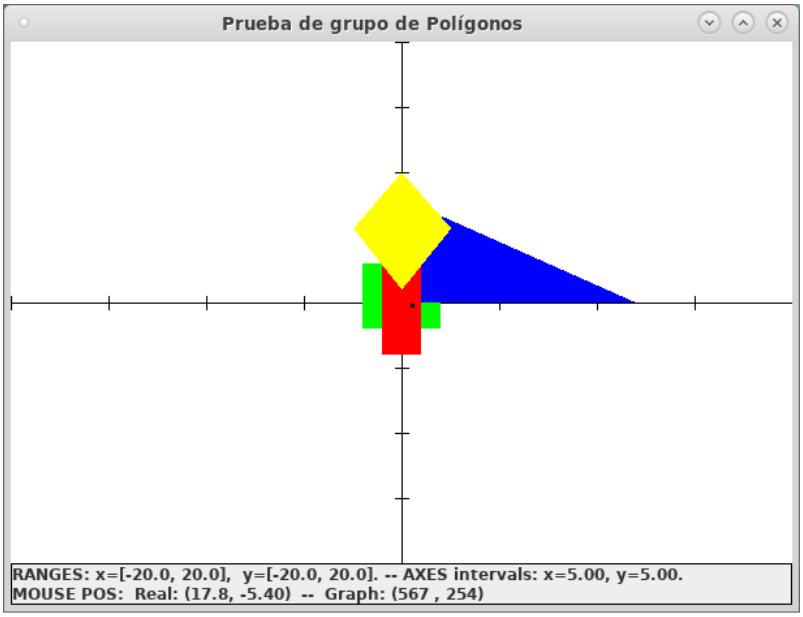


Figura 1: Grupo de polígonos.

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, en el grupo de polígonos se revisarán sus elementos por orden, de más arriba a más abajo (desde el frente hacia el fondo), 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, el de coordenadas `(0.6,0.8)` en concreto.

[1] Instalada y usada en la práctica 1 de PRG.

Para dar cuenta del comportamiento descrito, en aquella práctica se desarrolló una pequeña aplicación formada por las siguientes clases:

- 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`.

Las clases anteriores usaban la clase `Point`, puntos `(x, y)` en el plano cartesiano, desarrollada en una práctica anterior.

En esta práctica se van a volver a utilizar estas clases. En concreto:

- Las clases `Point` y `Polygon` se suponen ya implementadas y listas para su uso. En el material para la práctica se proporcionan `Point.class` y `Polygon.class`. En la sección 3 se describen los métodos de la clase `Polygon`.
- La clase `PolygonGroup` va a tener la misma interfaz de métodos, pero ahora se va a cambiar la estructura de sus objetos: si en la práctica de IIP se usaba un array para almacenar los polígonos que forman parte del grupo, ahora se va a usar una secuencia enlazada. Los métodos de la clase se deberán implementar en consecuencia a lo largo de esta práctica. Para desarrollar las secuencias enlazadas, se usará una clase `NodePol`, nodo cuyo dato es de tipo `Polygon`.
- Dado que la clase `Test7` usa únicamente los métodos públicos de `PolygonGroup`, y estos van a mantener el mismo perfil y significado, en esta práctica se va a poder reutilizar para realizar pruebas de los métodos, visualizando gráficamente su resultado, haciendo uso de la librería gráfica `Graph2D` (usada ya en la práctica 1). Así, la clase `Test7` se llama `Test5` y la única diferencia con `Test7` es que, para realizar la lectura validada de datos desde teclado, se utilizan los métodos de la clase `CorrectReading` (del paquete `utilPRG`) implementada en la práctica 4.

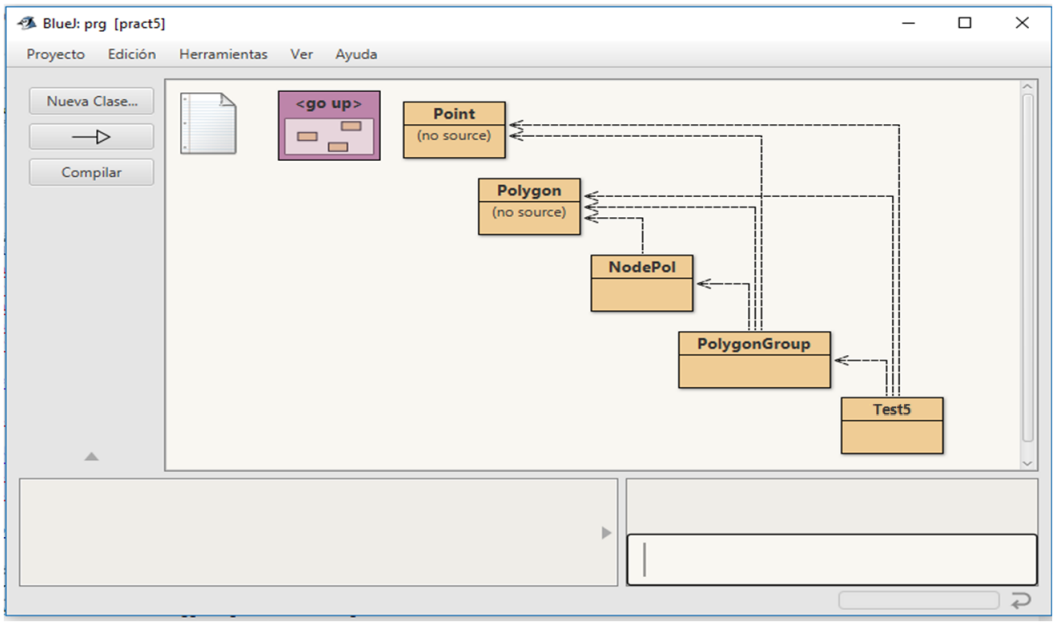


Figura 2: Clases del paquete `prg[pract5]`.

2.1. Actividad 1: preparación del paquete `pract5`

a) Descargar los ficheros `Point.class`, `Polygon.class`, `NodePol.java`, `PolygonGroup.java` y `Test5.java` disponibles en la carpeta de material para la práctica 5 de *PoliformaT*.

b) Abrir en *BlueJ* el proyecto de trabajo de la asignatura `prg` (desde un equipo propio o mediante conexión remota a un equipo del laboratorio) y crear un nuevo paquete `pract5`.

c) Agregar al paquete `pract5` las clases `NodePol.java`, `PolygonGroup.java` y `Test5.java` descargadas. Se podrán agregar las clases `Point.java` y `Polygon.java` que se desarrollaron en las prácticas 5 y 7 de IIP, respectivamente; si no se tuviera una versión validada de estas clases o si se prefiriese, en su lugar se puede copiar en la carpeta del paquete el código `.class` de ambas clases, que se habrá podido descargar de *PoliformaT*.

d) La librería gráfica (clase `Graph2D` del paquete `graph2D`) se facilita como una librería en el fichero `graphLib.jar` y su documentación se proporciona en el fichero `docGraph2D.zip` (en la carpeta *PRG:recursos/Laboratorio/Librería gráfica* de *PoliformaT*).

   Se debe cargar esta librería de la manera adecuada (indicando dónde se encuentra el fichero `jar`, que se habrá descargado previamente). Si la práctica se realiza en el laboratorio mediante conexión remota, ya la tendrás cargada y, por tanto, puedes obviar los pasos d.1) y d.2) descritos a continuación. Si la práctica se realiza en un equipo propio:

   d.1) Sitúa el fichero `graphLib.jar` en el proyecto de prácticas `prg` y, desde el menú *Preferencias/Librerías* de *BlueJ*, opción *Add File*, añade el fichero `graphLib.jar`. Tendrás que arrancar nuevamente *BlueJ* para que la librería se cargue.

   d.2) Descomprime el fichero `docGraph2D.zip` en el proyecto `prg` para poder consultar la documentación de la clase `Graph2D` (fichero `Graph2D.html`) cuando sea necesario.

3. Interfaz de la clase `Polygon`

Como ya se vio en la práctica 7 de IIP, 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.

Los métodos de la clase son los que se describen a continuación, y ya están implementados en la clase `Polygon.class` que se proporciona como material.

- Constructor `public Polygon(double[] x, double[] y)`: construye un `Polygon` a partir de un array `x` con las abscisas $\mathtt{x}_0$, $\mathtt{x}_1$, $\mathtt{x}_2$, ..., $\mathtt{x}_{n-1}$ de sus vértices, y un array `y` con las ordenadas $\mathtt{y}_0$, $\mathtt{y}_1$, $\mathtt{y}_2$, ..., $\mathtt{y}_{n-1}$ de sus vértices, siendo $n > 0$. Los vértices definen un polígono cuyos lados se extienden de un vértice al siguiente, y cerrándose en $(\mathtt{x}_0, \mathtt{y}_0)$. 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()`, `public double[] verticesY()` y `public Point[] vertices()` que, respectivamente, devuelven un array con las sucesivas abscisas de los vértices, un array con las sucesivas ordenadas de los vértices y un array con los sucesivos 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)`, 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`.

Notar que los métodos `verticesX()`, `verticesY()` y `getColor()` permiten obtener los datos de un polígono que necesita el método `fillPolygon` de la librería `Graph2D`, el cual dibuja un polígono en una ventana a partir de las abcsisas y ordenadas de sus vértices. Por ejemplo, en la figura 3 se muestra un fragmento de código, en el *CodePad* de *BlueJ*, que crea un triángulo azul y lo dibuja en una ventana gráfica.

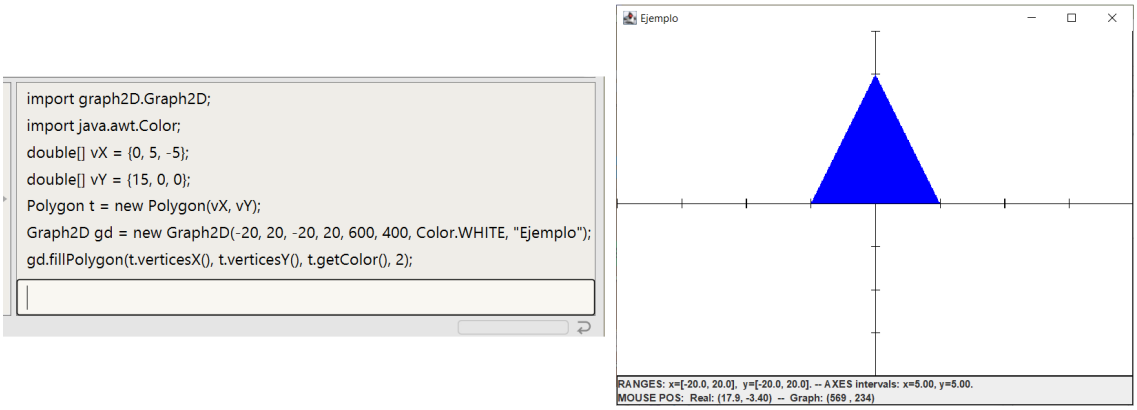


Figura 3: Ejemplo de creación de un polígono y posterior dibujo en una ventana gráfica.

4. La clase `NodePol`

La clase `NodePol` es análoga a la clase `NodeInt` vista en el tema 5, excepto que los datos deben ser de tipo `Polygon`. Se usa para implementar secuencias enlazadas de polígonos.

4.1. Actividad 2: declaración de atributos de la clase `NodePol`

Se debe completar la clase `NodePol` con la definición de sus atributos. Los métodos constructores, análogos a los de `NodeInt`, se han dejado ya resueltos.

5. La clase `PolygonGroup` implementada mediante secuencias enlazadas

Un `PolygonGroup` vendrá dado por la secuencia de polígonos que forman parte del grupo. Se implementará almacenando los polígonos en una secuencia enlazada.

Dado que una de las operaciones primordiales en la gestión del grupo, la de seleccionar un polígono mediante un clic, requiere buscar el polígono por orden de más arriba a más abajo en el grupo, lo más indicado es que la secuencia venga en dicho orden, de forma que el primero en la secuencia sea el polígono al frente del grupo, y el último el del fondo. Por este motivo, como cada nuevo polígono que se añada al grupo se debe superponer a los demás, se insertará en cabeza.

Para facilitar otras operaciones, como la llevar al fondo, es conveniente mantener en el grupo otra referencia al final de la secuencia. Así pues, la clase se define mediante los siguientes atributos (actividad 3):

- `front`: atributo privado de instancia de tipo `NodePol`, nodo en cabeza de la secuencia de polígonos.
- `back`: atributo privado de instancia de tipo `NodePol`, nodo final de la secuencia de polígonos.
- `size`: atributo privado de instancia de tipo `int`, la talla o número de polígonos en el grupo.

En la figura 4 se muestra un objeto `PolygonGroup` que corresponde al grupo de la figura 1.

Los métodos de la clase se deben implementar en las actividades 4, 5 y 6.

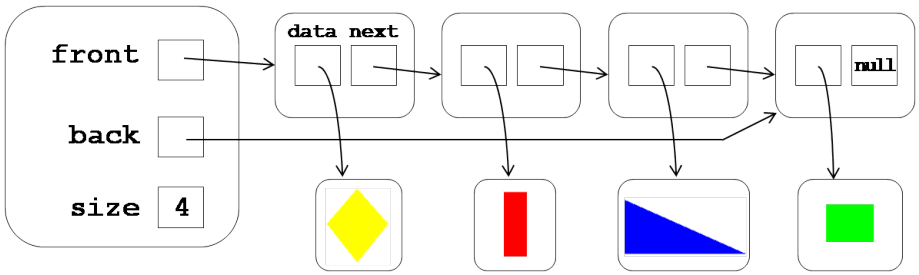


Figura 4: Un `PolygonGroup` de talla 4.

5.1. Actividad 3: declaración de los atributos de la clase `PolygonGroup`

Editar la clase `PolygonGroup.java` para completar la declaración de sus atributos según la descripción anterior.

5.2. Actividad 4: implementación y prueba de los métodos constructor `PolygonGroup`, `getSize`, `add` y `toArray`

Para completar la clase, en esta actividad se empezará implementando los métodos más sencillos.

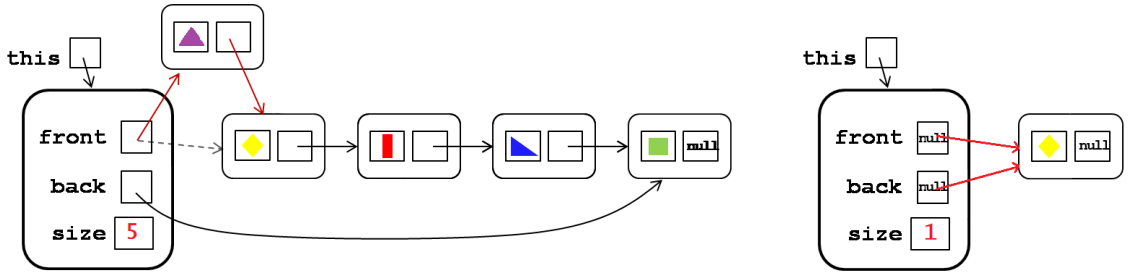
- Constructor `public PolygonGroup()`: construye un `PolygonGroup` vacío.

  La referencias `front` y `back` se dejarán a `null`, y la talla `size` a 0.

- Método `public int getSize()`, que devuelve la talla o número de polígonos en el grupo.

- Método `public void add(Polygon pol)`, que añade al frente del grupo, arriba del todo, el polígono `pol`.

  Notar que `pol` se debe insertar en un nuevo nodo en cabeza de la secuencia de polígonos (ver figura 5(a)). Si la inserción se hace en el grupo vacío, el nuevo nodo será al mismo tiempo el primero y el último de la secuencia, por lo que además de `front` y `size`, habrá que actualizar el atributo `back` (ver figura 5(b)).



(a) Si el grupo no está vacío (b) Si el grupo está vacío

Figura 5: Cambios que realiza el método `add` sobre un grupo de polígonos cuando se añade un nuevo polígono a la secuencia.

- 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, es decir, desde el fondo hasta el frente. El grupo no se modifica. Por ejemplo, para un grupo como el de la figura 4 debe devolver un array como el figura 6.

  Este método se ha dejado resuelto y resulta útil para realizar las pruebas del programa `Test5`. Notar que, como el orden de los polígonos en el array debe ser el inverso al de la secuencia, al copiar los sucesivos polígonos de la secuencia se recorre el array en sentido descendente.

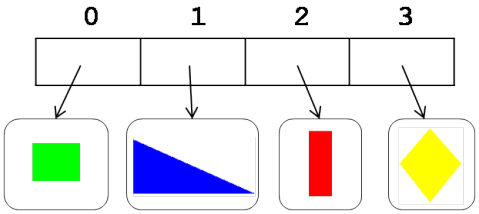


Figura 6: Array resultante de `toArray` sobre el grupo de la figura 4.

Para hacer unas pruebas preliminares se puede usar el programa `Test5`. Se debe examinar el método `main` y comprobar que realiza las siguientes acciones:

1. Crea un polígono verde, uno azul, uno rojo y uno amarillo, y los va añadiendo a un grupo inicialmente vacío.
2. Escribe en la salida estándar la talla del grupo.
3. Usando el método auxiliar `drawGroup`, definido en la propia clase `Test5`, obtiene con el método `toArray` un array con los polígonos del grupo en el orden en que se añadieron, y los dibuja en dicho orden en una ventana `Graph2D`.
4. Despliega un menú de opciones, entre las que se encuentra acabar el programa.

Así pues, si los métodos constructor, `add` y `getSize` son correctos, al ejecutar el `main` de `Test5`, se escribe en la salida estándar que el grupo tiene talla 4, y su representación gráfica es como la de la figura 1. Elegir la opción 0 del menú para terminar la prueba.

5.3. Actividad 5: implementación y prueba de los métodos `search` y `translate`

- Método auxiliar `private NodePol[] search(Point p)`, que busca en el grupo descendentemente, de más arriba a más abajo, el primer polígono que contiene a `p`. Devuelve un array de tipo `NodePol`, de forma que la componente 1 tiene una referencia al nodo en el que se ha encontrado el polígono, y la componente 0 una referencia al nodo anterior. Si el polígono encontrado es el primero de la secuencia, la componente 0 del resultado vale `null`. Si el polígono no se encuentra, la componente 1 del resultado vale `null`.

  Este método se usará en los métodos `remove`, `toFront`, `toBack` siguientes, para encontrar el nodo de la secuencia que contenga el polígono señalado por un punto `p`.

  Para implementarlo, en el cuerpo del método se podrá hacer en primer lugar una búsqueda del primer nodo cuyo dato sea un polígono que contenga a `p` (comprobándolo con el método `inside` de `Polygon`):

  ```
  NodePol aux = front, prevAux = null;
  while (aux != null && !aux.data.inside(p)) {
      prevAux = aux;
      aux = aux.next;
  }
  ```

  El método terminará devolviendo un array cuyas componentes sean `prevAux` y `aux`:

  ```
  NodePol[] s = new NodePol[2];
  s[0] = prevAux; s[1] = aux;
  return s;
  ```

- 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`. Este método no cambia la superposición relativa de los polígonos en el grupo.

  Este método se implementará buscando el primer nodo `mark` cuyo polígono contenga a `p`:

  ```
  NodePol[] s = this.search(p);
  NodePol mark = s[1];
  ```

  Si existe tal polígono (`mark != null`), entonces sólo hay que aplicar al polígono `mark.data` el método `translate` de `Polygon`, que modifica sus coordenadas según lo deseado.

Para probar estos métodos se podrá usar el programa `Test5`. El menú que despliega el método `main` permite seleccionar un polígono del grupo para aplicarle una operación, redibujando el grupo en el estado resultante; ello posibilita hacer las siguientes pruebas:

- Seleccionar mediante un clic el polígono del fondo, pedir que se traslade, y comprobar que se ha encontrado en el grupo este polígono y que cambia correctamente su posición en el plano.
- Repetir lo anterior para el polígono del frente.
- Repetir lo anterior para un polígono intermedio.
- Seleccionar mediante un clic un punto exterior a cualquier polígono del grupo, y comprobar que si se pide hacer un traslado del polígono seleccionado, entonces no se produce ningún cambio.

5.4. Actividad 6: implementación y prueba de los métodos `remove`, `toBack` y `toFront`

Una vez que se tiene que el método `search` es correcto, en esta actividad se deben completar el resto de métodos de la clase.

Todos estos métodos realizan una acción que suponen una alteración de la secuencia enlazada de polígonos del grupo. Reciben como parámetro un punto `p` y, mediante el método `search`, deben buscar el primer nodo en la secuencia cuyo dato sea un polígono que contenga a `p` para, en caso de que exista, eliminarlo, llevarlo al frente, o llevarlo al fondo del grupo. En lo que sigue, supondremos que todos ellos ejecutan en primer lugar el código que viene a continuación.

```
NodePol[] s = this.search(p);
NodePol prevMark = s[0], mark = s[1];
```

De tal manera que en `mark` se tendría la referencia al nodo, y en `prevMark` al nodo anterior.

- Método `public boolean remove(Point p)`, que elimina del grupo el polígono seleccionado mediante el punto `p`, devolviendo `true`. Si el punto `p` no está contenido en ningún polígono, el método devuelve `false`.

  Para implementar este método, si se hubiera encontrado un nodo cuyo dato fuera un polígono que contiene a `p`, se eliminará dicho nodo de la secuencia. Se distinguen dos casos:

  - El nodo a eliminar es el primero de la secuencia (el del frente), esto es, `mark` es igual a `front` (o, lo que es lo mismo, `prevMark` es `null`). En este caso, `front` pasará a ser el siguiente nodo de la secuencia (`null` si era el único), como se puede ver en el ejemplo de la figura 7 a).
  - El nodo a eliminar tiene un nodo que le precede, esto es, `mark` no es `front` (o `prevMark` es distinto de `null`). En este caso, habrá que actualizar el siguiente de `prevMark`, como en el ejemplo de la figura 7 b).

  Además, habrá que actualizar el atributo `size`. Y si el nodo a eliminar es el del fondo (`mark` es igual a `back`), habrá que actualizar también el atributo `back` (ver el ejemplo de la figura 8).

  El método no hace nada si no se encuentra ningún polígono que contenga al punto.

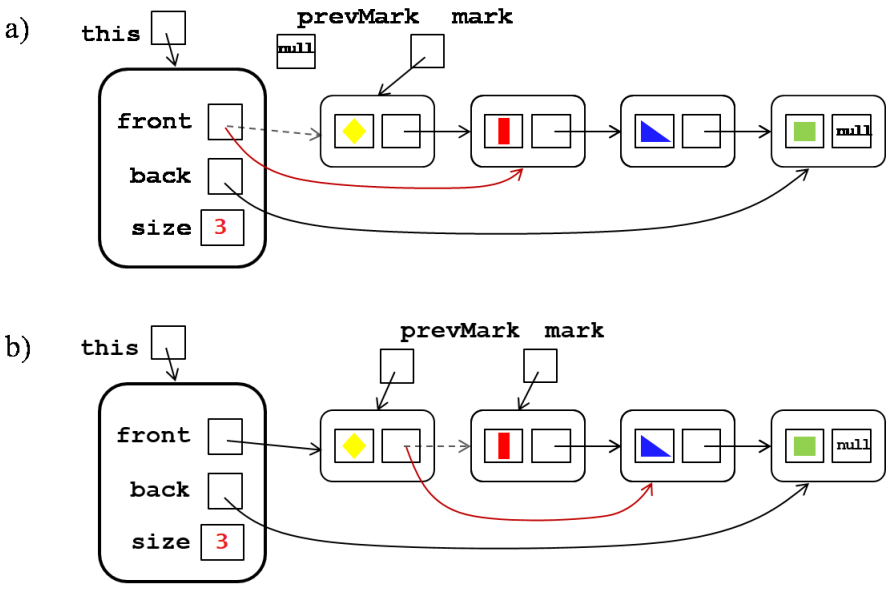


Figura 7: Cambios que realiza el método `remove` sobre el grupo de la figura 4 cuando se ha clicado el polígono del frente (polígono que se encuentra en el primer nodo de la secuencia) y cuando se ha clicado el rectángulo rojo (polígono en un nodo que tiene un anterior).

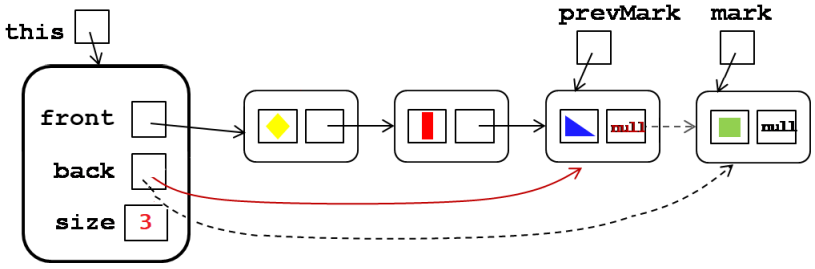


Figura 8: Cambios realizados por el método `remove` sobre el grupo de la figura 4 cuando se ha clicado el polígono del fondo (polígono que se encuentra en el último nodo de la secuencia).

Una vez implementado, se probará con el programa `Test5`, seleccionando y eliminando el polígono del frente, uno intermedio, el del fondo, y finalmente el único polígono que quede en el grupo (el programa escribe en la salida la talla del grupo tras cada eliminación). También se probará a clicar un punto exterior a cualquier polígono, comprobando que el método no hace nada.

- Método `public void toFront(Point p)`, que sitúa al frente del grupo, arriba del todo, el polígono seleccionado mediante el punto `p`. Si no hay ningún polígono que contenga a `p`, el método no hace nada.

  En el código que implementa este método, habrá que comprobar en primer lugar que se haya encontrado tal polígono, y que en ese caso no esté ya al frente del grupo; en caso contrario no hay que hacer nada.

  Hecha esta comprobación, y teniendo en cuenta que el nodo a trasladar tendrá uno que le preceda (no es el primero), hay que sacarlo de la secuencia y situarlo en cabeza (ver el ejemplo de la figura 9 a) y b)). Además, si el nodo trasladado fuera el del fondo, hay que actualizar también el atributo `back` (ver el ejemplo de la figura 10).

  Se repetirán las siguientes pruebas sobre el grupo del programa `Test5`: probar a traer al frente

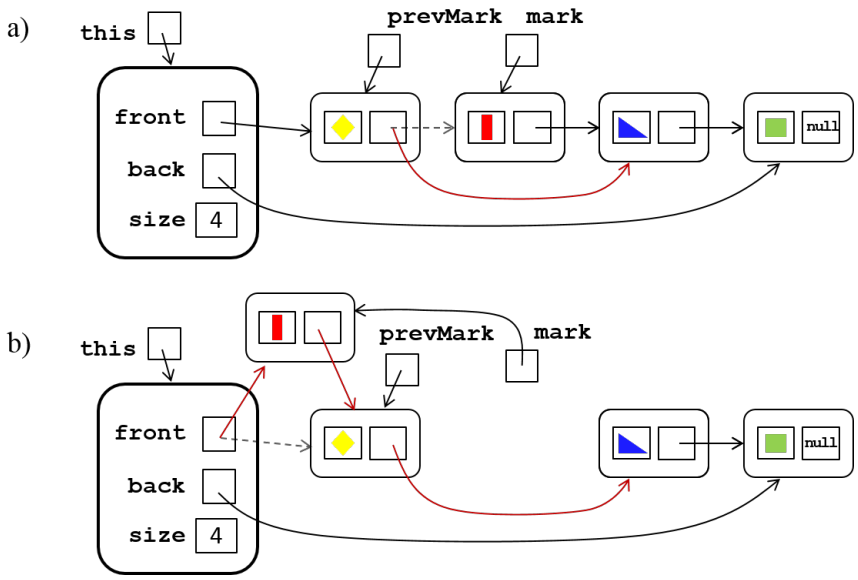


Figura 9: Cambios que realiza el método `toFront` sobre el grupo de la figura 4 cuando se ha clicado el rectángulo rojo.

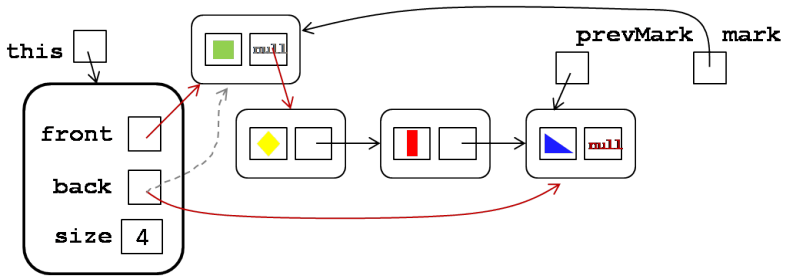


Figura 10: Cambios realizados por el método `toFront` sobre el grupo de la figura 4 cuando se ha clicado el polígono del fondo (polígono que se encuentra en el último nodo de la secuencia).

el polígono que ya está en el frente, traer al frente un polígono intermedio, y traer al frente el polígono del fondo; comprobar igualmente que el método no hace nada si se clica un punto exterior a cualquier polígono.

- Método `public void toBack(Point p)`, que sitúa al fondo del grupo, abajo del todo, el polígono seleccionado mediante el punto `p`. Si no hay ningún polígono que contenga a `p`, el método no hace nada.

  En la implementación de este método, si se hubiera encontrado un nodo cuyo polígono contuviera a `p`, y dicho nodo no estuviera ya en el fondo, entonces hay que resituar dicho nodo en la secuencia. A diferencia del método anterior, al sacar el nodo de la secuencia hay que distinguir si el nodo tiene o no un nodo que le precede (figuras 11 a) y 12 a) respectivamente). En cualquiera de estos casos, hay que acabar poniendo el nodo al final de la secuencia (figuras 11 b) y 12 b)).

  Con el programa `Test5` se harán pruebas de este método análogas a las realizadas para el método `toFront`.

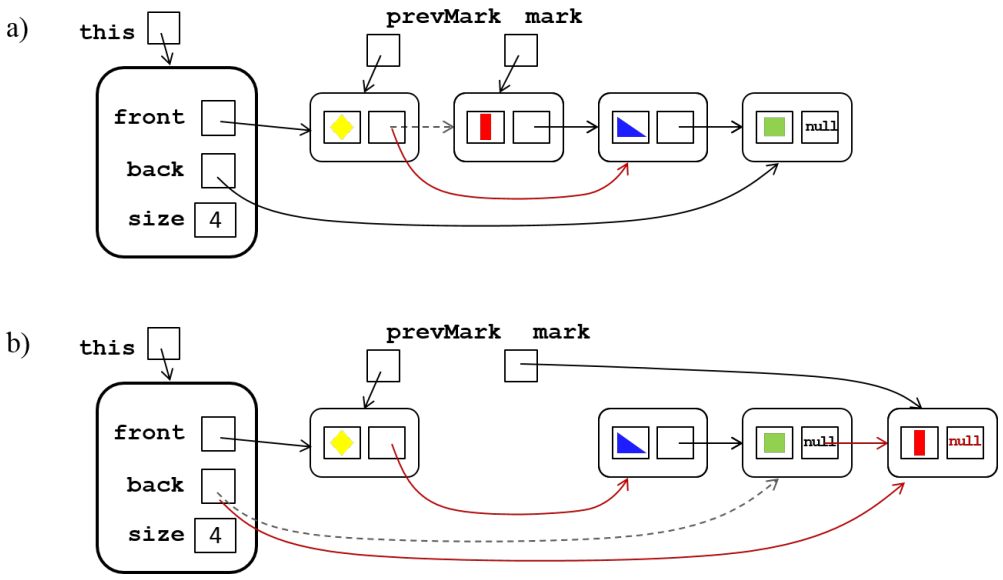


Figura 11: Cambios que realiza el método `toBack` sobre el grupo de la figura 4 cuando se ha clicado el rectángulo rojo (polígono en un nodo que tiene un anterior).

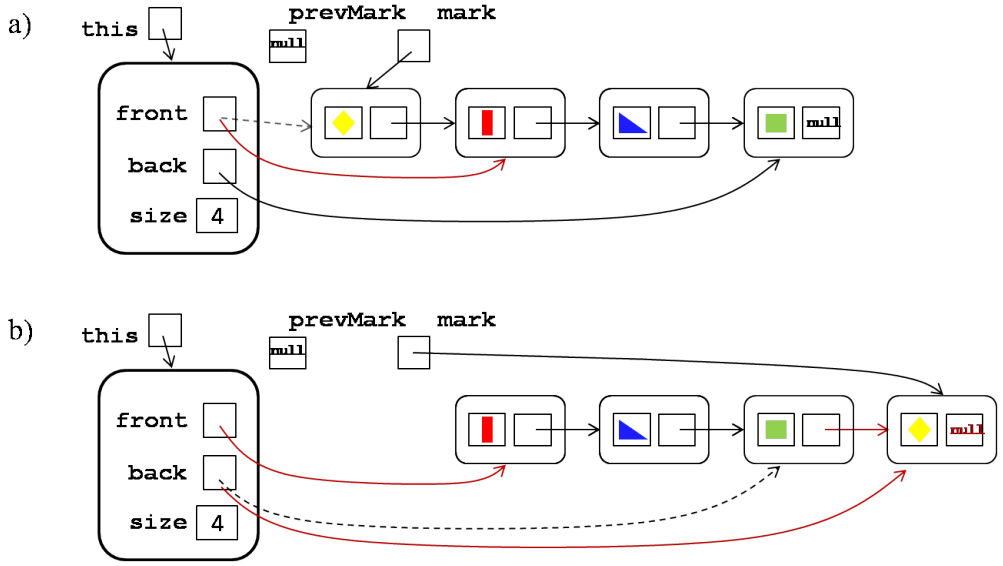


Figura 12: Cambios que realiza el método `toBack` sobre el grupo de la figura 4 cuando se ha clicado el polígono del frente (polígono que se encuentra en el primer nodo de la secuencia).

6. Validación de las clases

Cuando tu profesor lo considere conveniente, dejará disponibles en *PoliformaT* unas clases de prueba para validar tu código más allá de las pruebas preliminares realizadas en las actividades anteriores.

En general, para pasar el test 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.

6.1. Actividad 7: validación de las clases `NodePol` y `PolygonGroup`

Descargar los archivos correspondientes a las *Unit Test* sobre el directorio del paquete `pract5` y reabrir tu proyecto `prg` desde *BlueJ*.

Primero se hará la validación de la clase `NodePol` y, si estuviera correcta, se procederá a validar `PolygonGroup`.

Elegir 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. 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.

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*.