PRG (E.T.S. d'Enginyeria Informàtica)
Academic Year 2019-2020

Lab activity 5. Linked sequences: an application to the management of polygons in the two-dimensional space

(2 sessions)

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Contents

1 Objectives and previous work to the lab sessions

This practice is an exercise to develop a data structure, in which the linked sequences presented in the last subject of the subject will be used. The implementation of the methods of the developed class will give rise to exercise the basic algorithms on such sequences: insertion and deletion of data, searches, and other simple changes in the order of the data.

It is supposed that, prior to the work of the laboratory, the description of the problem and the organisation of classes proposed for its resolution have been carefully read.

The practice lasts two sessions in which the activities proposed in this bulletin will be developed. Activities 1 to 5 should be completed during the first session. Activities 6 and 7 during the second session.

2 Description of the problem

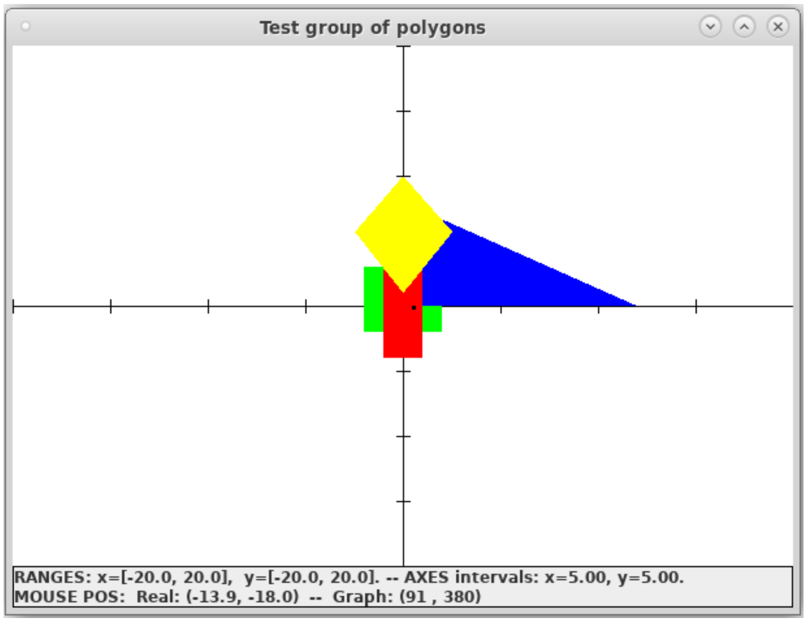
In practice 7 of IIP, the problem of managing a group of polygons, which were supposed to be arranged on a plane, was discussed and described as follows:

Each polygon is given by the sequence of its vertices (points in the plane), and it is of a certain fill colour (colour of the surface of the polygon) so that, from said data, it can be drawn in a space of drawing like those in the graphic library `Graph2D` [1] of the package `graph2D`. A polygon can move on the plane and change the fill colour.

A group of polygons is, as its name suggests, a grouping of polygons to which new elements can be added, existing elements can be deleted, or all the elements of the group can be treated in solidarity. You can also select a polygon of the group to work with it individually, such as moving it or changing the colour.

The typical behaviour in the graphic applications in which a group of figures is handled, polygons in this case, is that they are superimposed by the order in which they are added to the group. Thus, if there is an overlap between the surfaces of the polygons, below is the oldest, and above all, the most recent one. For example, the drawing in figure 1 graphically represents a group of polygons in which a green rectangle, a blue triangle, a red rectangle and a yellow quadrilateral have been added in order.

This order can be changed by selecting one polygon to send it to the bottom or to bring it to the front (the top), etc.



```
RANGES: x=[-20.0, 20.0],  y=[-20.0, 20.0]. -- AXES intervals: x=5.00, y=5.00.
MOUSE POS:  Real: (-13.9, -18.0)  --  Graph: (91 , 380)
```

Figure 1: Group of polygons.

When a group of polygons has been represented graphically, the way in which a polygon of the group can be selected is by clicking on a point of its surface that is visible, that is, it is not hidden by other superimposed polygons. In the graphic part of this practice the point of the click will be detected, so that in the group of figures, known that point, the indicated polygon can be accessed. To do this, in the group of polygons, their elements will be checked in order, from top to bottom (from the front to the bottom), looking for the first one that contains that point: this is the polygon to be selected, given that it is superimposed to the rest of polygons that contain the same point. In the example of the figure 1 we can see how, to indicate the red rectangle, one of the visible points of its surface, the one of `(0.6,0.8)` in particular, has been clicked.

[1] Installed and used in lab practice 1 of PRG.

To account for the behaviour described, in that practice a small application was developed consisting of the following classes:

- Class `Polygon`. For representing and managing polygons in the plane.
- Class `PolygonGroup`. For representing and managing groups of polygons in the plane.
- Class `Test7`. Test program that creates a group of figures and allows you to try different actions on the group. To help verify the effect of these actions, this program graphically visualises the results, using the `Graph2D` library.

The previous classes used the class `Point`, points `(x, y)` in the Cartesian plane, developed in a previous practice.

In this lab practice we are going to use these classes again. In particular:

- The classes `Point` and `Polygon` are assumed already implemented and ready for use. In the material for the practice, `Point.class` and `Polygon.class` are provided.

  In the section 3 the methods of the class `Polygon` are described.
- The class `PolygonGroup` will have the same interface of methods, but now the structure of its objects will be changed: if in the practice of IIP an array was used to store the polygons that are part of the group, now a linked sequence will be used. The methods of the class should be implemented accordingly throughout this practice. To implement the linked sequences, we will use a class `NodePol`, a node whose data is of type `Polygon`.
- Since the class `Test7` uses only the public methods of `PolygonGroup`, and these will maintain the same profile and meaning, in this practice it will be able to be reused to test the methods, graphically visualising their result, using the graphical library `Graph2D` (already used in lab practice 1). Thus, the `Test7` class is renamed to `Test5`, and the only difference with the `Test7` class is that, to perform the validated reading of data from the keyboard, the methods of the `CorrectReading` class (in `utilPRG` package, implemented in lab activity 4) are used.

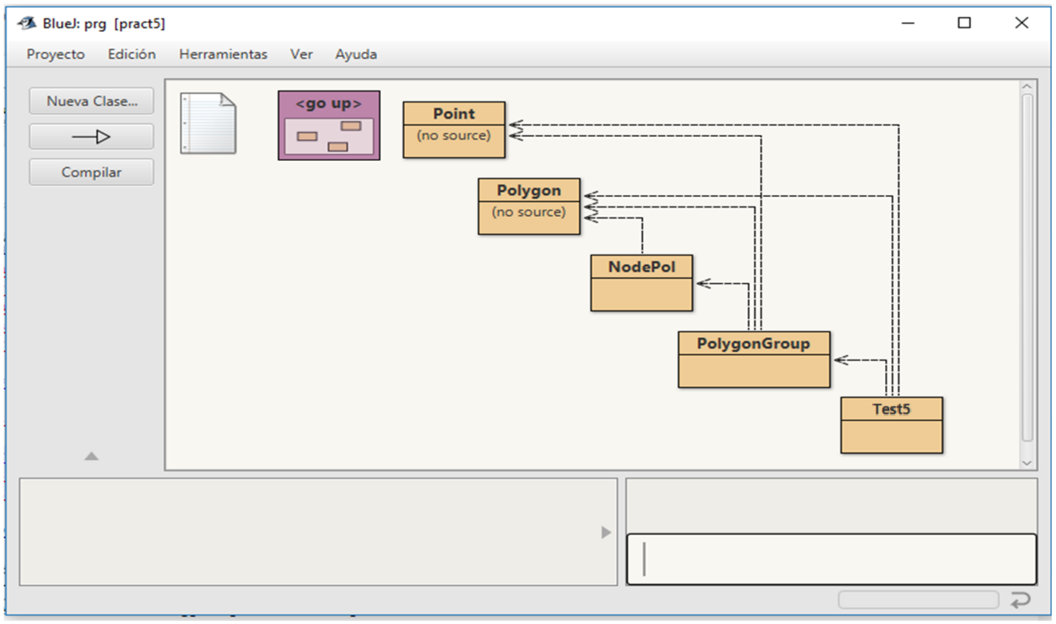


Figure 2: Classes of package `prg[pract5]`.

2.1 Activity 1: setup of the package `pract5`

a) Download the files `Point.class`, `Polygon.class`, `NodePol.java`, `PolygonGroup.java` and `Test5.java` available in the material folder for the lab practice 5 of *PoliformaT*.

b) Open the project *BlueJ* for (`prg`) and create a new package named `pract5`.

c) Add to the package `pract5` the classes `NodePol.java`, `PolygonGroup.java` and `Test5.java` downloaded. You can add the classes `Point.java` and `Polygon.java` that were developed in practices 5 and 7 of IIP, respectively; if you do not have a validated version of these classes or if you prefer, instead you can copy the code `.class` of both classes in the package folder, which could have been downloaded from *PoliformaT*.

d) The graphical library is provided as the class `Graph2D` in the package `graph2D`. This package is contained in the JAR file `graphLib.jar` and its documentation in the file `docGraph2D.zip` (availables in the folder *PRG:recursos/Laboratorio/Librería gráfica* of *PoliformaT*).

   This library should properly loaded indicating where the JAR file is located in the file system. If you use the laboratory virtual desktops (by using remote connection), you will have it loaded and, therefore, you can skip steps d.1) and d.2) described below. If you use your own computer:

   d.1) Set the file `graphLib.jar` in the *BlueJ* project `prg`, and add it in *Preferencias/Librerías* of *BlueJ*.

   d.2) Extract the contents of the file `docGraph2D.zip` in the folder of the project `prg` in order to have access to the documentation when needed, in particular to file `Graph2D.html`.

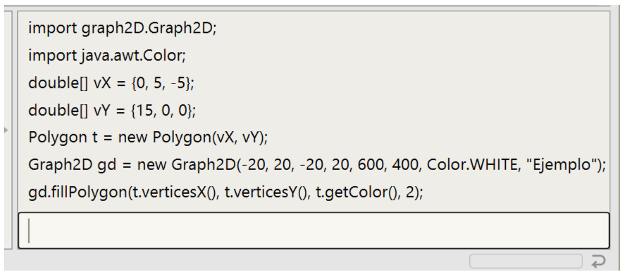
3 Interface of the class `Polygon`

As it was seen in the lab practice 7 of IIP, a `Polygon` is defined by a sequence of n vertices, $v_0, v_1, \ldots, v_{n-1}$, so that their sides are the segments $\overline{v_0v_1}, \overline{v_1v_2}, \ldots, \overline{v_{n-2}v_{n-1}}, \overline{v_{n-1}v_0}$. The polygon has a fill colour.

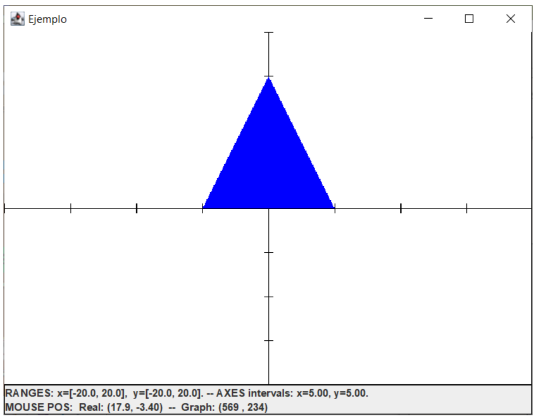
The methods of the class are described below, and are already implemented in the class `Polygon.class` that is provided as material.

- Constructor `public Polygon(double[] x, double[] y)`: creates a `Polygon` from an array `x` with the abscissas $\mathtt{x}_0$, $\mathtt{x}_1$, $\mathtt{x}_2$, ..., $\mathtt{x}_{n-1}$ of their vertices, and an array `y` with the ordinates $\mathtt{y}_0$, $\mathtt{y}_1$, $\mathtt{y}_2$, ..., $\mathtt{y}_{n-1}$ of their vertices, being $n > 0$. The vertices define a polygon whose sides extend from one vertex to the next, and closing in $(\mathtt{x}_0, \mathtt{y}_0)$. By default, the polygon is blue (`Color.BLUE`).
- Methods `public Color getColor()` and `public void setColor(Color nC)`, colour getter and colour modifier respectively.
- Methods `public double[] verticesX()` and `public double[] verticesY()` that, respectively, return an array with successive abscissas of the vertices, and an array with the successive ordinates of the vertices.
- Method `public void translate(double incX, double incY)`, which translates the vertices of the polygon: `incX` on the X axis, `incY` on the Y axis.
- Method `public double perimeter()`, which returns the perimeter of the polygon.
- Method `public boolean inside(Point p)`, check if the `Point p` is inside the polygon. If the point is inside the polygon it returns `true`, and if the point is outside the polygon it returns `false`.

Note that the methods `verticesX()`, `verticesY()` and `getColor()` allow you to obtain the data of a polygon that the method `fillPolygon` of the library `Graph2D` needs, which draws a polygon in a window from the abscissas and ordinates of its vertices. For example, in the figure 3 a fragment of code, in the CodePad of BlueJ, is shown that creates a blue triangle and draws it in a graphic window.



```
import graph2D.Graph2D;
import java.awt.Color;
double[] vX = {0, 5, -5};
double[] vY = {15, 0, 0};
Polygon t = new Polygon(vX, vY);
Graph2D gd = new Graph2D(-20, 20, -20, 20, 600, 400, Color.WHITE, "Ejemplo");
gd.fillPolygon(t.verticesX(), t.verticesY(), t.getColor(), 2);
```



```
RANGES: x=[-20.0, 20.0],  y=[-20.0, 20.0]. -- AXES intervals: x=5.00, y=5.00.
MOUSE POS:  Real: (17.9, -3.40)  --  Graph: (569 , 234)
```

Figure 3: Example of creating a polygon and drawing it in a graphic window.

4 La clase `NodePol`

The class `NodePol` is analogous to the class `NodeInt` seen in unit 5, except that the data must be of type `Polygon`. It is used to implement linked sequences of polygons.

4.1 Activity 2: declaration of attributes of the class `NodePol`

The class `NodePol` must be completed with the definition of its attributes. The constructor methods, analogous to those of class `NodeInt`, have been left completed.

5 The class `PolygonGroup` implemented by means of linked sequences

A `PolygonGroup` will be given by the sequence of polygons that are part of the group. It will be implemented by storing the polygons in a linked sequence.

Given that one of the most important operations in the management of the group is selecting a polygon by means of a click, it is necessary to search the polygon from the top to the bottom in the group, the best strategy is to start from the first node in the sequence because the first node contains the polygon in front of the group, and the last node the one that is in the background. For this reason, as each new polygon that is added to the group must be superimposed on the others, it will be inserted in the head.

To facilitate other operations, such as pushing one polygon to the back, it is convenient to keep another reference in the group pointing to the end of the sequence. Thus, the class is defined by the following attributes (activity 3):

- `front`: private instance attribute of type `NodePol`, node in the head of the sequence of polygons.
- `back`: private instance attribute of type `NodePol`, node at the end of the sequence of polygons.
- `size`: private instance attribute of type `int`, the size of the sequence, i.e. the number of polygons in the group.

Figure 4 shows an object of the class `PolygonGroup` corresponding to the group in Figure 1.

Methods of this class must be implemented in activities 4, 5 and 6.

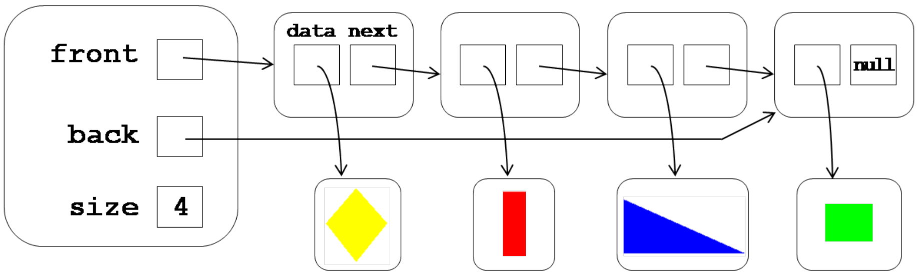


Figure 4: A PolygonGroup with 4 polygons.

5.1 Activity 3: declaration of the attributes of the class PolygonGroup

Edit the class PolygonGroup.java to complete the declaration of its attributes according to the previous description.

5.2 Activity 4: implementation and testing of the constructors of the class PolygonGroup, and methods add, getSize and toArray

To complete the class, this activity will start by implementing the simplest methods.

- Constructor public PolygonGroup(): creates an empty PolygonGroup.

  References front and back must be set to null, and size set to 0.

- Method public int getSize(), that returns the size of the group, i.e. then number of polygons in the group.

- Method public void add(Polygon pol), that adds in the front of the group the polygon pol.

  Note that pol must be inserted into a new node at the head of the polygon sequence (see figure 5(a)). If the insertion is made in the empty group, the new node will be both the first and the last of the sequence, so in addition to front and size, the attribute back has to be updated too (see figure 5(b)).

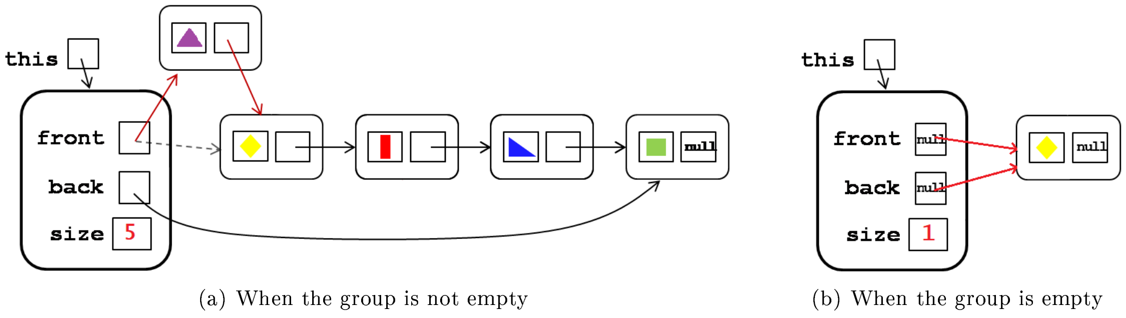


Figure 5: Changes that performs the add method on a group of polygons where a new polygon is added to the sequence.

- Method public Polygon[] toArray(), which returns an array of length equal to the size of the group with the sequence of polygons of the group, in order from the lowest to the highest, that is, from the bottom to the front. The group does not change. For example, for a group like the one in the figure 4 you must return an array like the figure 6.

  This method has been left solved and is useful for testing the Test5 program. Note that, since the order of the polygons in the array must be the inverse of the sequence, when copying the successive polygons of the sequence, the array is traversed in a downward direction.

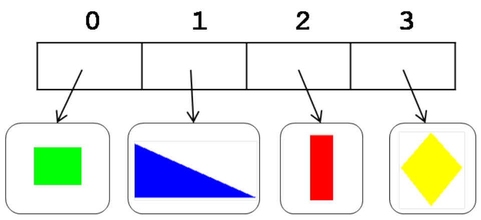


Figure 6: Array returned by the method `toArray` from the group shown in Figure 4.

To do some preliminary tests you can use the program `Test5`. You should examine the `main` method and verify that it performs the following actions:

1. Create a green, a blue, a red and a yellow polygon, and add them to an initially empty group.
2. Write the size of the group in the standard output.
3. Using the auxiliary method `drawGroup` defined in the own class `Test5`, it obtains with the method `toArray` an array with the polygons of the group in the order in which they were added, and draws them in such order in a window `Graph2D`.
4. Displays a menu of options, among which is to finish the program.

So, if the constructor methods, `add` and `getSize` are correct, when executing the `main` of `Test5`, it is written in the standard output that the group has size 4, and its graphic representation is like that of the figure 1. Choose option 0 from the menu to finish the test.

5.3 Activity 5: implementation and test of methods `search` and `translate`

- Auxiliary method `private NodePol [] search(Point p)`, which searches the group down, from top to bottom, for the first polygon containing `p`. Returns an array of type `NodePol`, so that component 1 has a reference to the node where the polygon was found, and component 0 a reference to the previous node. If the polygon found is the first one in the sequence, the 0 component of the result is `null`. If the polygon is not found, the 1 component of the result is `null`.

  This method will be used in the following methods `remove`, `toFront`, `toBack`, to find the node of the sequence containing the polygon indicated by a point `p`.

  To implement it, in the body of the method you can first make a search for the first node whose data is a polygon containing `p` (checking it with the `inside` method of `Polygon`):

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

  The method will end returning an array containing `prevAux` and `aux`:

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

- Method `public void translate(Point p, double incX, double incY)`, which translates the selected polygon by the point `p`. The abscissas of their vertices are increased by `incX`, and the ordinates by `incY`. This method does not change the relative overlap of the polygons in the group.

This method will be implemented looking for the first node `mark` whose polygon contains `p`:

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

If there is such a polygon (`mark != Null`), then you just have to apply the `mark.data` polygon to the `translate` of `Polygon`, which modifies its coordinates as desired.

To test these methods, the program `Test5` can be used. The menu that displays the `main` method allows you to select a polygon of the group to apply an operation, redrawing the group in the resulting state; this makes it possible to do the following tests:

- Select the background polygon with a click, request that it be moved, and check that this polygon has been found in the group and that it correctly changes its position in the plane.
- Repeat the above for the polygon of the front.
- Repeat the above for an intermediate polygon.
- Select by a click a point outside any polygon of the group, and check that if a transfer of the selected polygon is requested, then no change occurs.

5.4 Activity 6: implementation and test of methods `remove`, `toBack` and `toFront`

Once you have that the `search` method is correct, in this activity the other methods of the class must be completed.

All these methods perform an action that involves an alteration of the linked sequence of a group polygons. They receive as parameter a point `p` and, by means of the method `search`, they must search for the first node in the sequence whose data is a polygon containing `p`, for, in case it exists, eliminate it, bring it to the front, or push it to the bottom of the group. From now on, we will assume that all these methods execute the code that comes next at the beginning.

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

In such a way that `mark` would have the reference to the node, and in `prevMark` to the previous node.

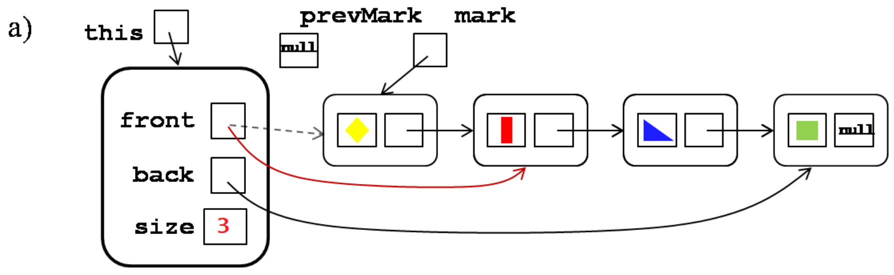
- Method `public boolean remove(Point p)`, which removes the selected polygon from the group by means of the `p` point and it returns `true`. If none of the polygons of the group are selected by the point `p`, this method returns `false`.

  To implement this method, if a node whose data was a polygon containing `p` was found, that node will be eliminated from the sequence. There are two different cases:

  - The node to be erased is the first one in the sequence (the frontmost one); i.e., `mark` has the same value than `front` (or, in an equivalent manner, `prevMark` is `null`). In this case, `front` will become the next node in the sequence (`null` if it was the only one present), as it can be seen in the example in Figure 7 a).
  - The node to be erased has a preceding node, i.e., `mark` is not `front` (or `prevMark` is different from `null`). In that case, the next to `prevMark` must be updated, as in the example shown in Figure 7 b).

  Additionally, the `size` attribute must be updated. And when the node to be erased is that of the bottom (`mark` is equal to `back`), the `back` attribute must be updated (see example in Figure 8).

  The method does nothing if no polygon containing the point is found.



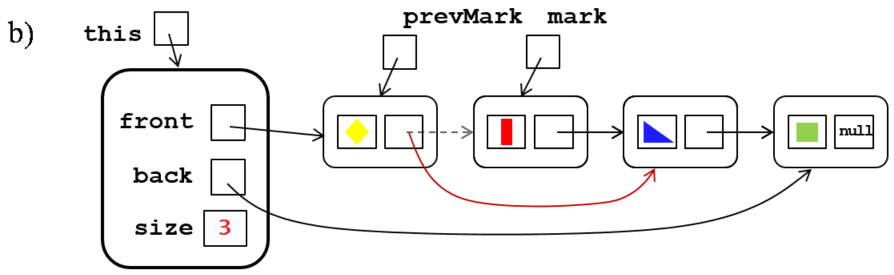


Figure 7: Changes performed by the method `remove` on the group in Figure 4 when the front polygon is clicked (polygon that is the first one in the sequence) and when the red rectangle is clicked (polygon that has a previous node).

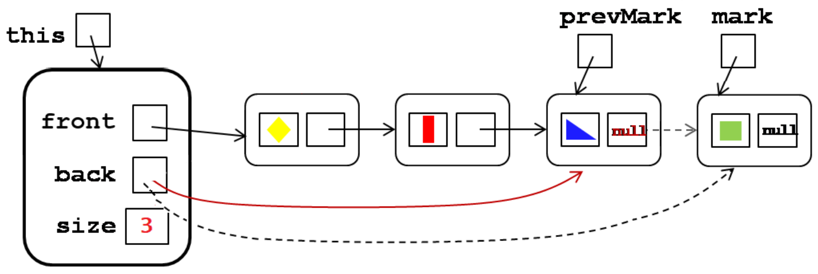


Figure 8: Changes performed by the method `remove` on the group in Figure 4 when the bottom polygon is clicked (polygon that is the last one in the sequence).

Once implemented, it will be tested with the program `Test5`, selecting and eliminating the polygon from the front, an intermediate polygon, the bottom polygon, and finally the only polygon that remains in the group (the program writes the size of the group after each elimination). It will also be tried to click a point outside any polygon, checking that the method does not do anything.

- Method `public void toFront(Point p)`, which places the polygon selected by the point `p` at the top of the group. If there is no polygon containing `p`, the method does nothing.

  In the code that implements this method, it will be necessary to check that such a polygon has been found, and in that case it is not already in front of the group; otherwise, nothing should be done.

  Done this check, and taking into account that the node to be transferred will have one that precedes it (it is not the first), it must be removed from the sequence and placed at the top (see the example of Figure 9 a) and b)). In addition, if the transferred node is the one in the background, the attribute `back` must also be updated (see the example in the figure 10).

  The following tests on the group of the program `Test5` will be repeated: try to bring to the front the polygon that is already in the front, bring forward an intermediate polygon, and

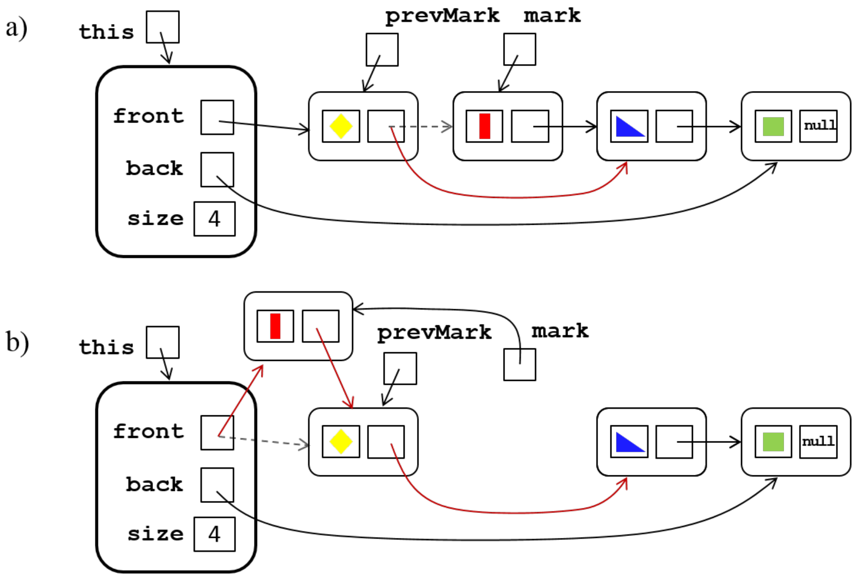


Figure 9: Changes performed by the method `toFront` on the group show by Figure 4 when the red rectangle was selected.

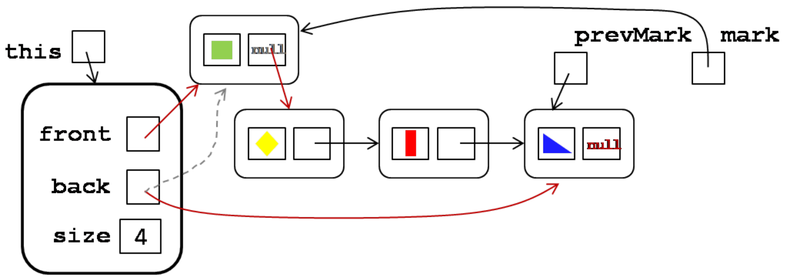


Figure 10: Changes performed by the method `toFront` on the group show by Figure 4 when the polygon at the bottom was selected, i.e. the last one in the linked sequenced.

bring to the front the polygon of the bottom; check also that the method does not do anything if you click on a point outside any polygon.

- Method `public void toBack (Point p)`, which places the polygon selected by the point `p` at the bottom of the group. If there is no polygon containing `p`, the method does nothing.

  In the implementation of this method, if a node whose polygon contained `p` had been found, and that node was not already in the background, then that node must be re-positioned in the sequence. Unlike the previous method, when removing the node from the sequence, it is necessary to distinguish whether or not the node has a preceding node (Figures 11 a) and 12 a) respectively). In any of these cases, it is necessary to end up putting the node at the end of the sequence (Figures 11 b) and 12 b)).

  With the `Test5` program, analogous tests to those made for the `toFront` method must be made for this method.

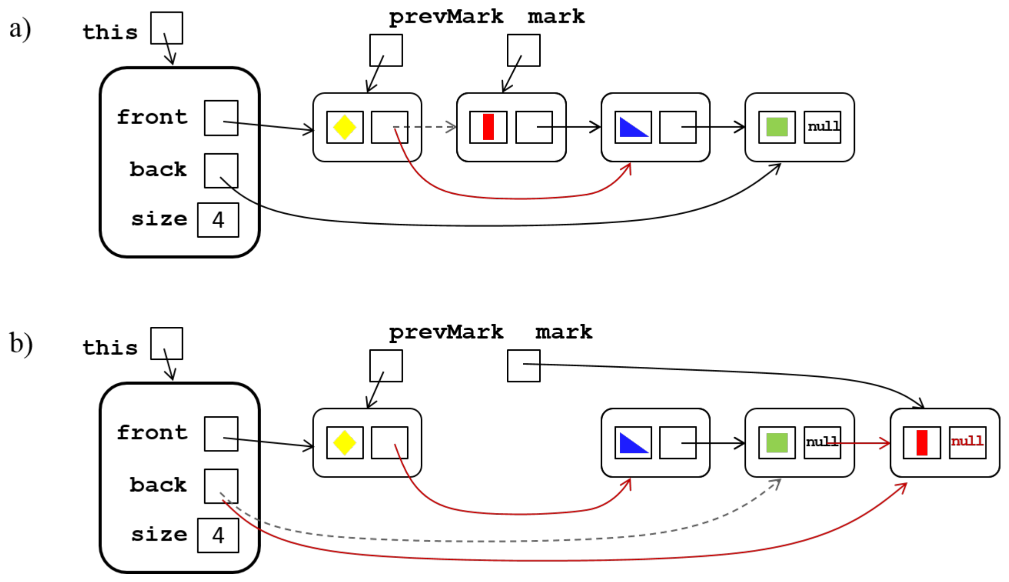


Figure 11: Changes performed by the method toBack on the group shown by Figure 4 when the red rectangle has been selected. It is a polygon on a node that has a previous node.

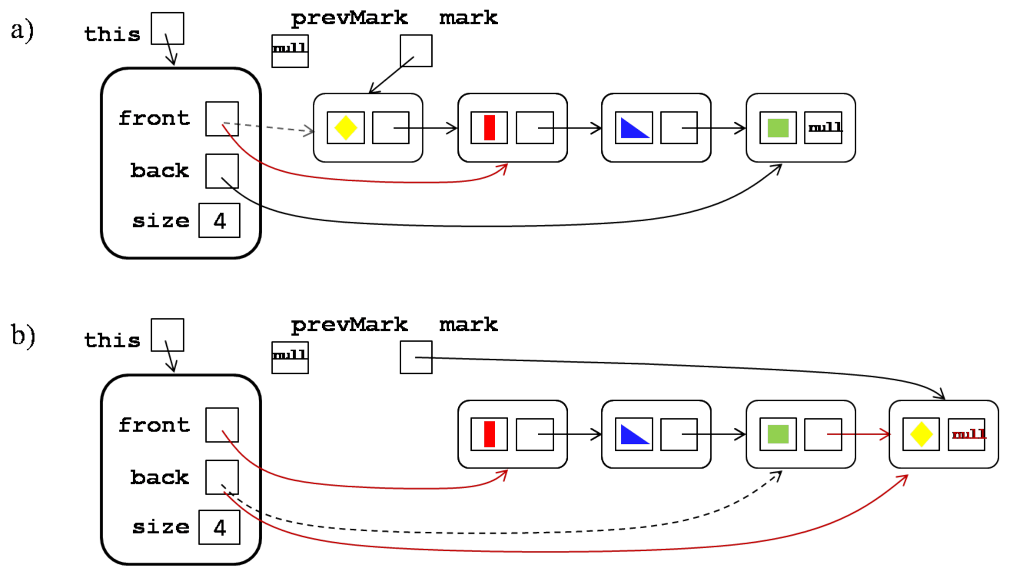


Figure 12: Changes performed by the method toBack on the group shown by Figure 4 when the first polygon in the sequence has been selected. The first one in the linked sequence.

6 Validation of the classes

When your teacher deems it convenient, he will leave available in *PoliformaT* some test classes to validate your code beyond the preliminary tests carried out in the previous activities.

In general, to pass the test correctly, you must make sure that you use the same identifiers of attributes and methods proposed in this document and in the documentation of the files `.java` that you are provided, strictly following the characteristics on modifier methods and parameters proposed in the header of the methods.

6.1 Activity 7: validation of classes `NodePol` and `PolygonGroup`

Download the files corresponding to the *Unit Test* on the package directory `pract5` and reopen your project `prg` from *BlueJ*.

First the validation of the class `NodePol` will be done and, if it is correct, the validation of the class `PolygonGroup`.

Choose the option *Test All* from the context menu that appears when you click with the right mouse button on the icon of the class *Unit Test*. As always, a set of tests will be executed on the implemented methods of the corresponding class, comparing expected results with those actually obtained. As in previous practices, if the methods are correct, they will be marked with the symbol ✓ (green colour) in the *Test Results* window of *BlueJ*. On the other hand, if one of the methods does not work correctly, then it will be marked by the symbol `X`. If you select any of the lines marked with `X`, a more descriptive message about the possible cause of error is displayed at the bottom of the window.

If, after correcting errors and recompiling the class, the icon of the *Unit Test* is checked, then close and reopen the project *BlueJ*.