IIP (E.T.S. de Ingeniería Informática)
Year 2014-2015

Lab activity 2. Objects, classes, and programs.
The BlueJ environment

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Contents

1 Context and previous work

In the academic framework, this is the second lab activity of the course, and its main objective is the definition and use of simple classes by using the *BlueJ* integrated development environment (IDE). With this activity the student must get familiar with the basic aspects of *BlueJ*, and employ it from now as working environment for the next lab activities. The student must be able to employ the IDE for:

- creating a *BlueJ* project with the classes that are given at the beginning of the activity,
- designing and adding a new class to the project,
- editing some of the existing classes,
- creating objects and executing methods on them by using the *BlueJ* Object Bench,
- evaluate expressions by using the *BlueJ* Code Pad,
- generate automatically the documentation for the project,
- validate the correctness of the class or any method of it, by using the *BlueJ* Debugger.

To obtain the maximum performance during the lab sessions, the student must read this report before coming to the lab.

2 Developing a *BlueJ* project

As was indicated in lab activity 1 (*Introduction: Linux, Java, and BlueJ*), a *BlueJ* project consists of a set of related classes. *BlueJ* not only allows to develop the project (create, compile, execute, debug, document), but *allows to interact with any method (attribute or object) of any of the classes of the project* as well. The aim of this activity is getting the student familiar with the *BlueJ* environment.

2.1 Calling *BlueJ* and creating a project

As explained in the previous lab activity, *BlueJ* can be called from the system menu (`Aplicaciones - Otras - BlueJ 3.1.1`) or from the command line, with or without arguments:

```
bluej &
bluej projectname &
```

Notice that the second line is equivalent to call *BlueJ* without arguments and then, by using the option `Project - Open` from the menu, selecting the project `projectname`. If you want to open in *BlueJ* an existing Java application **which was not developed with *BlueJ***, then call *BlueJ* without arguments and open the existing application (the directory where its classes are stored) with the menu option `Project - Open Non BlueJ...`.

Activity #1

1. Create a subdirectory `labact2` in the `iip` working directory.
2. Download the `.java` and `.class` files available in the PoliformaT folder `Recursos/Laboratorio/Práctica 2/Code` in the `labact2` directory.

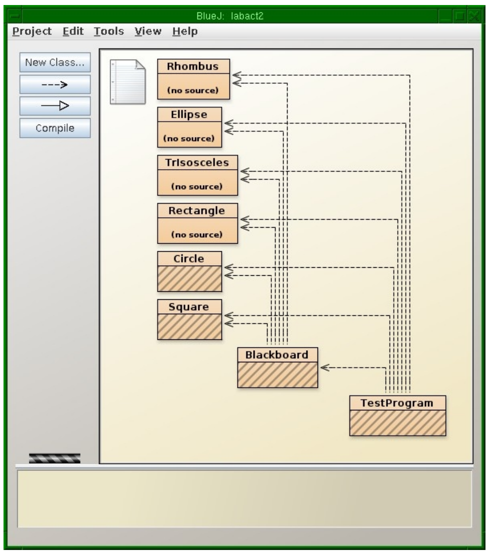


Figure 1: Project `labact2`.

3. Call *BlueJ* without arguments.

4. Open the folder `labact2` with the option `Project - Open Non BlueJ...`.

In *BlueJ* main window appear the icons of each class of the project (Figure 1). Notice that the icons associated to `.class` files appear with the text `(no source)`, and those associated to `.java` files appear with strips when they have not been compiled. The arrows show the use relations between classes.

2.2 Edition and compilation

For editing a class, in our case `Circle`, the option `Open Editor` from the menu class can be selected. As an alternative, double click on the class icon can be used.

With respect to how compiling a class, there are different ways:

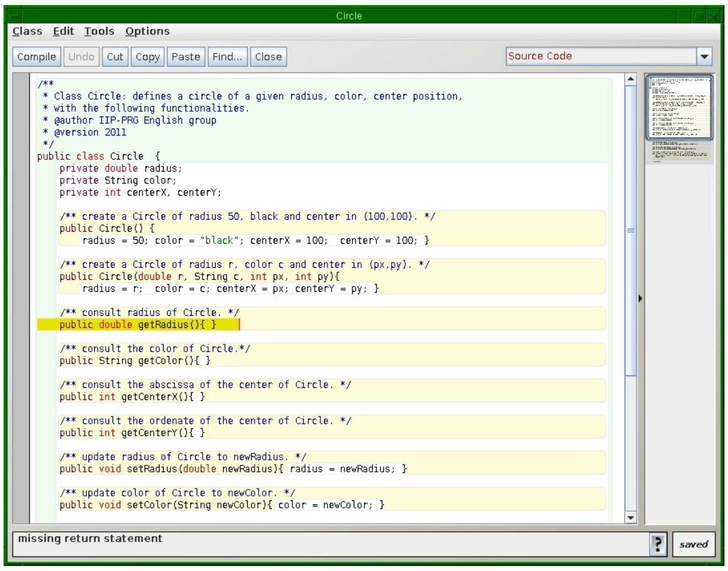
- from the editor;
- *when all the project is going to be compiled*, from the tool bar of the main window of *BlueJ*.

If the compiler detects any error, the corresponding line appears shadowed, and a message error will appear in the *information zone* as well (bottom part of the screen) (Figure 2). In this case, when the button with the symbol "`?`" (right part of the information zone) is clicked, more information can be obtained about the detected error.

Activity #2

1. Fill in all the consultor methods for the class `Circle`.

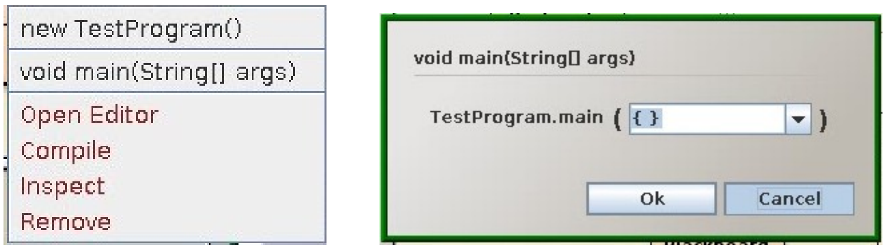
2. Compile all the classes, correct the compilation errors and check what happens in the central part of the *BlueJ* main window.



```
/**
 * Class Circle: defines a circle of a given radius, color, center position,
 * with the following functionalities.
 * @author IIP-PRG English group
 * @version 2011
 */
public class Circle  {
    private double radius;
    private String color;
    private int centerX, centerY;

    /** create a Circle of radius 50, black and center in (100,100). */
    public Circle() {
        radius = 50; color = "black"; centerX = 100;  centerY = 100; }

    /** create a Circle of radius r, color c and center in (px,py). */
    public Circle(double r, String c, int px, int py){
        radius = r;  color = c; centerX = px; centerY = py; }

    /** consult radius of Circle. */
    public double getRadius(){ }

    /** consult the color of Circle.*/
    public String getColor(){ }

    /** consult the abscissa of the center of Circle. */
    public int getCenterX(){ }

    /** consult the ordenate of the center of Circle. */
    public int getCenterY(){ }

    /** update radius of Circle to newRadius. */
    public void setRadius(double newRadius){ radius = newRadius; }

    /** update color of Circle to newColor. */
    public void setColor(String newColor){ color = newColor; }
```

```
missing return statement
```

Figure 2: Compilation of class `Circle`.



(a) Class menu. (b) `main` method call.

Figure 3: Execution of the class `TestProgram`.

2.3 Class execution: calling `main`

As was said before, when a class is clicked by using the mouse right button, the class menu appears. From the operations that shows this menu, the call to the `main` method must be highlighted, since *allows to execute a class from its menu.*

In the Figure 3(a) the pop-up menu for the class `TestProgram` is shown, and in the Figure 3(b) the call window for the `main` method of the `TestProgram` class.

Activity #3

1. Execute the `main` method of the class `TestProgram`. The result must be similar to that that is shown in the Figure 4.

2.4 *Tools* option. Documentation generation

From the different utilities for the `Tools` option, the most important are:

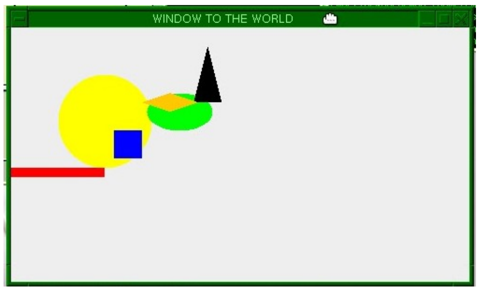
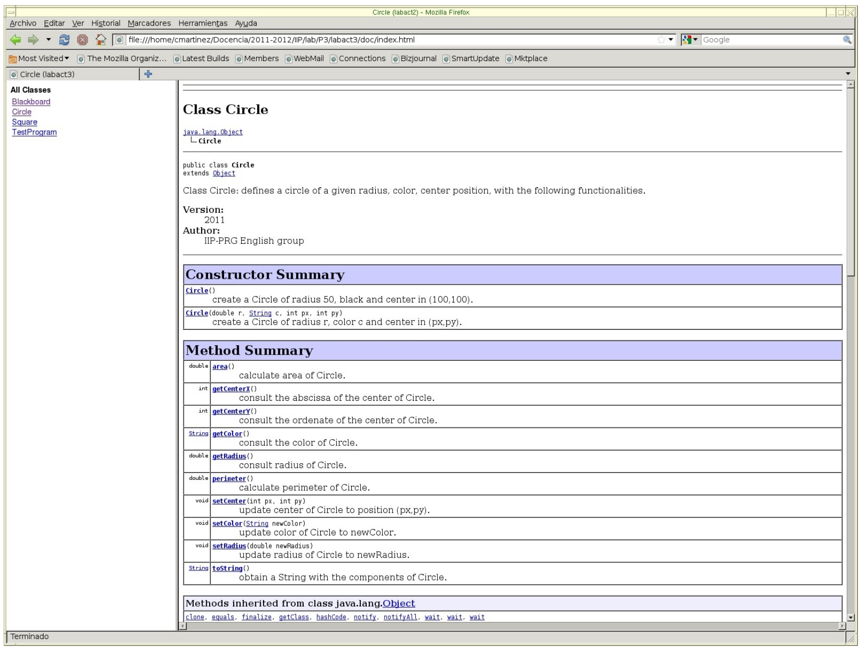


Figure 4: Result for the execution of the class `TestProgram`.


Circle (labact2) - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
file:///home/cmartinez/Docencia/2011-2012/IIP/lab/P3/labact3/doc/index.html
Google
Most Visited The Mozilla Organiz... Latest Builds Members WebMail Connections BizJournal SmartUpdate Mktplace
Circle (labact3)
Terminado

All Classes
Blackboard
Circle
Square
TestProgram

Class Circle

```
java.lang.Object
  └ Circle
```

```
public class Circle
extends Object
```

Class Circle: defines a circle of a given radius, color, center position, with the following functionalities.

Version:
2011
Author:
IIP-PRG English group

Constructor Summary
`Circle()` create a Circle of radius 50, black and center in (100,100).
`Circle(double r, String c, int px, int py)` create a Circle of radius r, color c and center in (px,py).

Method Summary	
double	`area()` calculate area of Circle.
int	`getCenterX()` consult the abscissa of the center of Circle.
int	`getCenterY()` consult the ordenate of the center of Circle.
String	`getColor()` consult the color of Circle.
double	`getRadius()` consult radius of Circle.
double	`perimeter()` calculate perimeter of Circle.
void	`setCenter(int px, int py)` update center of Circle to position (px,py).
void	`setColor(String newColor)` update color of Circle to newColor.
void	`setRadius(double newRadius)` update radius of Circle to newRadius.
String	`toString()` obtain a String with the components of Circle.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Figure 5: Documentation for project `labact2`.

- `Project Documentation`, generates the subdirectory `doc` with the documentation on the classes in `html` format.
- `Preferences...`, in its suboption `Misc` allows to define where the class documentation is situated.

Activity #4

1. Generate the project documentation and consult it. The result must be similar to that which appears in Figure 5.

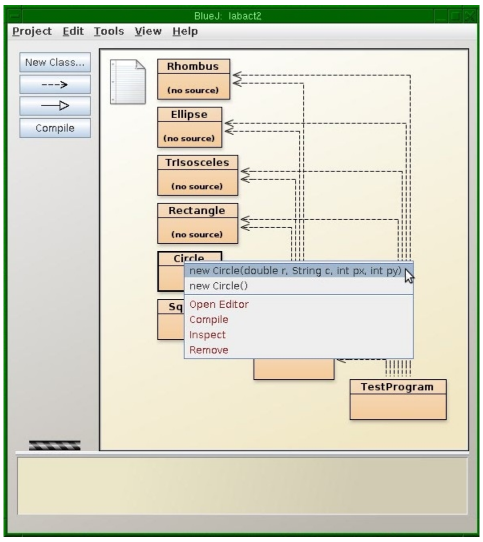


Figure 6: Menu for the class `Circle`.

2.5 *Help* option

The utility `Java Class Libraries` allows to access the Java documentation. In the labs, this help is in `http://docs.oracle.com/javase/8/docs/api/`, that must be checked and, in case it is necessary, modify the option `Tools - Preferences...`. The teacher will indicate in the lab session where it is situated the local version of the documentation, and the student *BlueJ* preferences could be changed to this local version.

Other utilities from `Help` allow to consult the *BlueJ* manual, as well as accessing its web site `www.bluej.org`.

3 Using the Object Bench

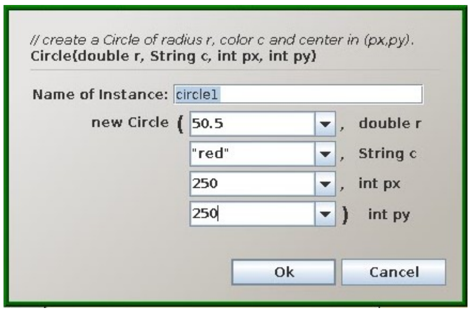
One of the most interesting features of the *BlueJ* IDE is its capability to interact with isolated objects from any class and execute the methods defined on these objects; thus, the functionality of a class can be checked before writing any application that uses it.

3.1 Class operations

To access to the possible operations, the class icon must be selected and clicked with the mouse right button. Then, it appears a list of the constructors of the class, along with other operations permitted by the IDE - such as removing or compiling the class (Figure 6).

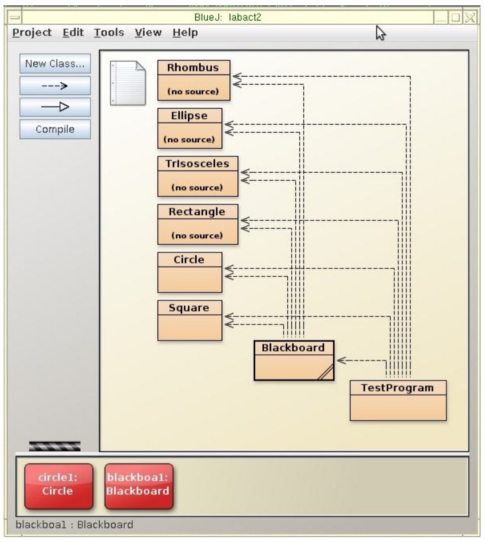
3.2 Object creation

To create an object, a constructor must be selected from the pop-up menu, and follow the dialogue box which gets opened (Figure 7(a)). Specifically, a name for the object is demanded. When the object is created, it appears in the bottom left corner of the *BlueJ* main window, in a zone known as *Object Bench* (Figure 7(b)).



```
// create a Circle of radius r, color c and center in (px,py).
Circle(double r, String c, int px, int py)

Name of Instance: circle1
new Circle ( 50.5  , double r
             "red" , String c
             250   , int px
             250   ) int py

Ok   Cancel
```

(a) Creating a `Circle` object.



(b) Object Bench.

Figure 7: Creating an object in the Object Bench of *BlueJ*.

3.3 Object methods execution

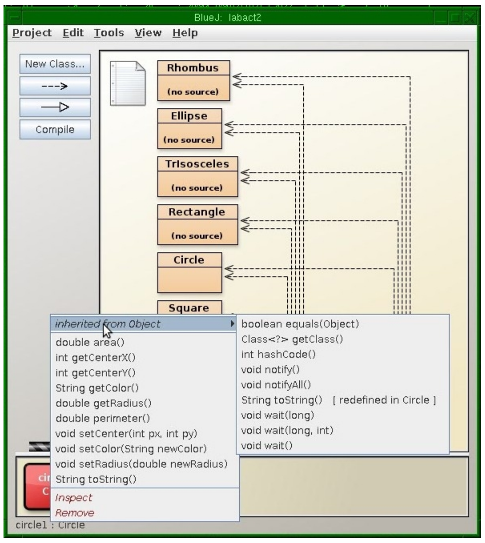
When the created object is clicked with the right button, the methods that can be executed on it can be accessed (Figure 8(a)). To execute any of them it must be selected. When the object inherits methods from other classes, they appear in submenus.

3.4 Watching the state of the object

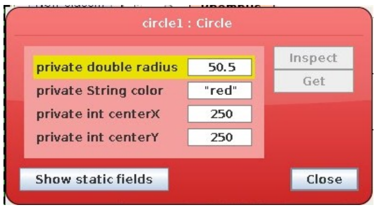
To debug the designed methods the `Inspect` option can be used. This operation allows to know the values of the attributes of the objects (Figure 8(b)).

Activity #5

1. Create an object of the `Circle` class with radius 50.5, color red, and centre in $(250, 250)$.
2. Inspect the values of the created object.
3. Execute the `toString()` method defined in the `Circle` class on the created object.
4. Modify the radius of `Circle` to 30.0.
5. Execute again the method `toString()`.
6. Create an object of the class `Blackboard` with title “Drawing” and size 500×500.
7. Inspect the values of the attributes of the created object.
8. Add the `Circle` objecto to the `Blackboard`.



(a) Menu for a `Circle` object.


circle1 : Circle

private double radius	50.5
private String color	"red"
private int centerX	250
private int centerY	250

(b) Inspecting the `Circle` object.

Figure 8: Menu and inspection of the state of an object.

4 Using the Code Pad

The *BlueJ* Code Pad is situated on the right bottom corner, besides the Object Bench (Figure 9). In case it is not shown, it can be shown by selecting the option `View - Show Code Pad`.

This zone can be used to input Java expressions or instructions:

- An instruction can be written, that is executed when *Enter* is pressed; for example, a variable can be declared and initialised; the variable will be available during the rest of the session

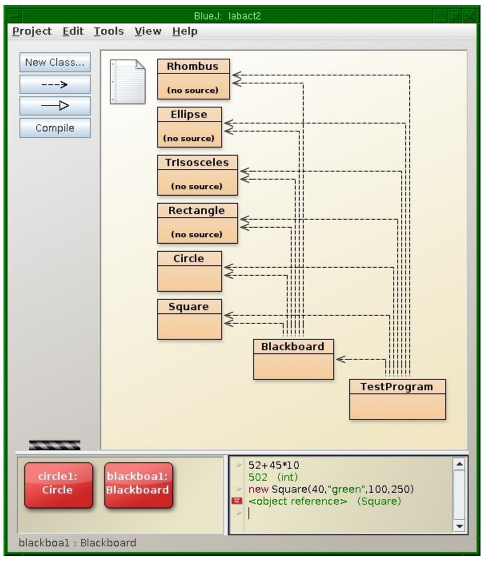
- An expression can be written; after pressing *Enter*, it will be evaluated and its final value shown, along its datatype (between parenthesis); objects from the Object Bench can be used as well

If the written code causes any compilation or execution error, the corresponding error message is shown.

Some results of expressions are objects instead of primitive values. In this case, the object is shown as a reference to an object (`<object reference>`), followed by the type of the object, and a small icon which represents the object is shown by the result line. That icon can be used to work with the result object, since it can be dragged to the Object Bench. In this case, the object will be situated on the Object Bench, where it will be available for future method calls (by using its pop-up menu or by using the Code Pad).

Activity #6

1. What are the results of evaluating the following expressions in the *BlueJ* Code Pad?



```
52+45*10
502   (int)
new Square(40,"green",100,250)
<object reference>   (Square)
```

Figure 9: *BlueJ* Code Pad

1	`99 + 3`	6	`9/2`
2	`8%3`	7	`9.0/2.0`
3	`-8%3`	8	`9/2.0`
4	`Math.sqrt(121)`	9	`9/(double)2`
5	`Math.sqrt(-5)`	10	`9/0`

2. Which of the following expressions return `true`?

1	`!(4<5)`
2	`!false`
3	`(2 > 2) || ((4 == 4) && (1 < 0))`
4	`(2 > 2) || (4 == 4) && (1 < 0)`
5	`(34 != 33) && !false`

3. Declare and init three variables `h`, `m`, and `s`, which respectively represent hours, minutes, and seconds (in 24 hour format) and write an expression that checks whether it is a correct hour or not.

4. Create an object of the `Circle` class on the Object Bench (by using the class pop-up menu, as described previously). Give to it the name `circleA`.

5. Write on the Code Pad an expression that returns the radius of `circleA`.

6. Create a new `Circle` object in the Code Pad and drag its icon to the Object Bench.

5 Using the Debugger

The *BlueJ* debugger is a simple tool, but very useful when validating the functionality of a class (i.e., its `main` method) or any other method of the class. Basically, it allows to:

- watch the execution of any method, i.e., *obtaining its trace for a given set of values*, maybe step by step, or maybe only for some of the code lines;
- inspect the call sequence associated to the call of the method on execution;
- check pass of parameters and the values of the local variables of the method on execution.

To obtain this, the debugger has the following functionalities:

1. **Establish breakpoints**. The execution state of a method can be viewed only when its execution is stopped in a line of its code. The Debugger provides this functionality to allow to stop the execution of the method in a specific line, that is, *establishes breakpoints*.

   In *BlueJ*, breakpoints are established in the *breakpoint area* of the editor, on the left of the text. When clicking in this area by a code line selected for stopping the execution, a small stop sign appears, which marks a breakpoint in that line. When the execution of a method arrives to a line with this mark, the execution gets interrupted and subsequently appear:

   (a) *the editor window*, in which the next line to the breakpoint line appears highlighted, since is the *next line to be executed*;

   (b) *the debugger window*; with the information items and buttons that are presented below.

2. **Step-by-step execution**. When the execution is stopped, it can be restarted step-by-step (instruction by instruction), which allows to follow the code and watch how the execution progresses (i.e., allows to *make a trace* of the code).

   To perform a step-by-step execution in *BlueJ* it is enough with clicking repeatedly on the `Step` button of the Debugger window. Each click produces the execution of an only code line, and then the execution stops again.

   To finish this process and return to normal execution, the breakpoint must be erased by clicking on it, and then the button `Continue` on the Debugger window must be clicked.

3. **Inspection of variables and pass of parameters**.

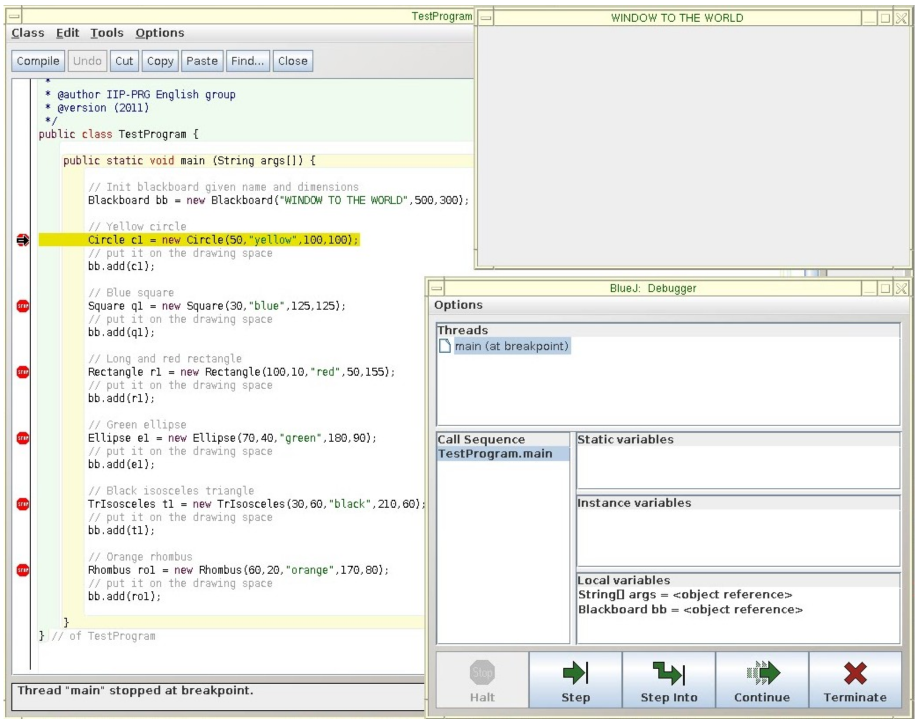
   By watching the *BlueJ* Debugger window, we can observe the call sequence associated to the method on execution, check the pass of parameters, and inspect the values that have its local variables.

   To inspect any variable or parameter you must double click on it on the Debugger window.

Activity #7

1. In the `main` method of the `TestProgram` class, create a breakpoint in the lines in which the objects of type `Circle`, `Square`, `Rectangle`, `Ellipse`, `TrIsosceles`, and `Rhombus` are created.
2. Execute the `main` method and observe what happens when, after reaching the breakpoint, the `Step` button of the Debugger window is clicked.
3. To inspect the variables of the `main` method, double click on them in the Debugger window.

In the Figure 10 the *BlueJ* Debugger is shown in two different situations during the execution of the `TestProgram` class.



```
 *
 * @author IIP-PRG English group
 * @version (2011)
 */
public class TestProgram {

    public static void main (String args[]) {

        // Init blackboard given name and dimensions
        Blackboard bb = new Blackboard("WINDOW TO THE WORLD",500,300);

        // Yellow circle
        Circle c1 = new Circle(50,"yellow",100,100);
        // put it on the drawing space
        bb.add(c1);

        // Blue square
        Square q1 = new Square(30,"blue",125,125);
        // put it on the drawing space
        bb.add(q1);

        // Long and red rectangle
        Rectangle r1 = new Rectangle(100,10,"red",50,155);
        // put it on the drawing space
        bb.add(r1);

        // Green ellipse
        Ellipse e1 = new Ellipse(70,40,"green",180,90);
        // put it on the drawing space
        bb.add(e1);

        // Black isosceles triangle
        TrIsosceles t1 = new TrIsosceles(30,60,"black",210,60);
        // put it on the drawing space
        bb.add(t1);

        // Orange rhombus
        Rhombus ro1 = new Rhombus(60,20,"orange",170,80);
        // put it on the drawing space
        bb.add(ro1);

    }
} // of TestProgram
```

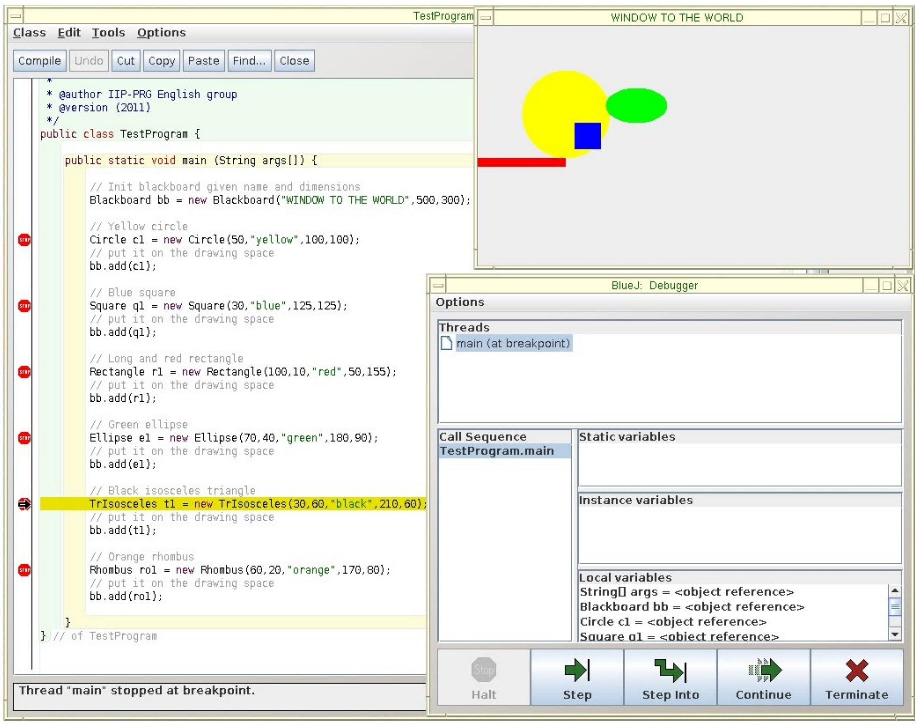
Thread "main" stopped at breakpoint.

Threads: main (at breakpoint)

Call Sequence: TestProgram.main

Local variables:
String[] args = <object reference>
Blackboard bb = <object reference>

(a) First breakpoint.



```
 *
 * @author IIP-PRG English group
 * @version (2011)
 */
public class TestProgram {

    public static void main (String args[]) {

        // Init blackboard given name and dimensions
        Blackboard bb = new Blackboard("WINDOW TO THE WORLD",500,300);

        // Yellow circle
        Circle c1 = new Circle(50,"yellow",100,100);
        // put it on the drawing space
        bb.add(c1);

        // Blue square
        Square q1 = new Square(30,"blue",125,125);
        // put it on the drawing space
        bb.add(q1);

        // Long and red rectangle
        Rectangle r1 = new Rectangle(100,10,"red",50,155);
        // put it on the drawing space
        bb.add(r1);

        // Green ellipse
        Ellipse e1 = new Ellipse(70,40,"green",180,90);
        // put it on the drawing space
        bb.add(e1);

        // Black isosceles triangle
        TrIsosceles t1 = new TrIsosceles(30,60,"black",210,60);
        // put it on the drawing space
        bb.add(t1);

        // Orange rhombus
        Rhombus ro1 = new Rhombus(60,20,"orange",170,80);
        // put it on the drawing space
        bb.add(ro1);

    }
} // of TestProgram
```

Thread "main" stopped at breakpoint.

Threads: main (at breakpoint)

Call Sequence: TestProgram.main

Local variables:
String[] args = <object reference>
Blackboard bb = <object reference>
Circle c1 = <object reference>
Square q1 = <object reference>

(b) Penultimate breakpoint.

Figure 10: *BlueJ* Debugger.