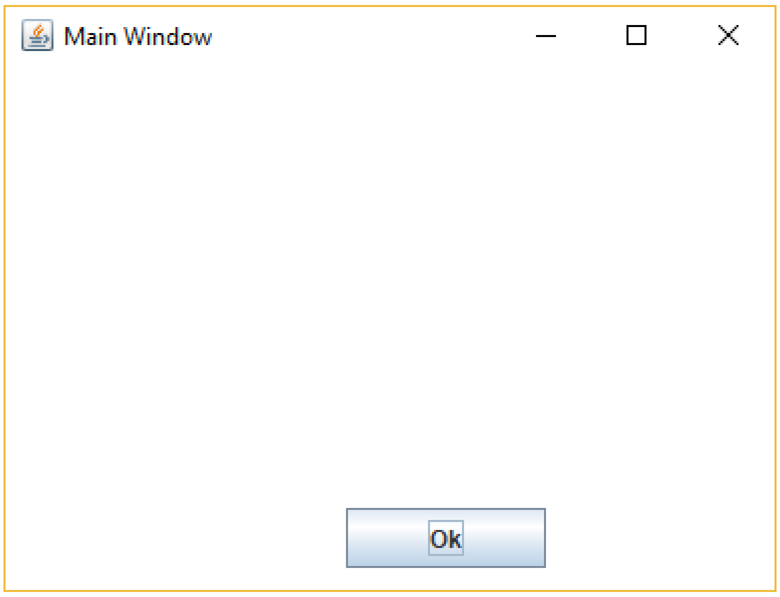
Lab 1

Creating an application with graphic user interface

1. Introduction

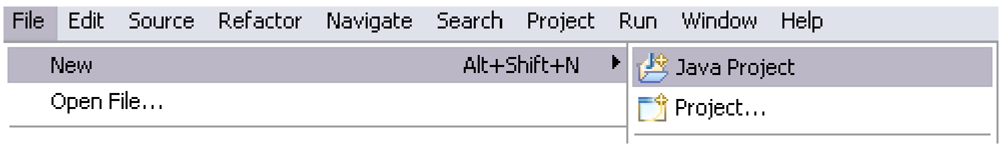
Windowed applications are the most usual programs we use in computers. These applications base their user interface in the typical windows provided by the underlying operating systems (MacOS, Windows and XWindows –the graphical server for Linux and Unix).
In this first lab we will create a very simple application that shows a window and an "Ok" button.



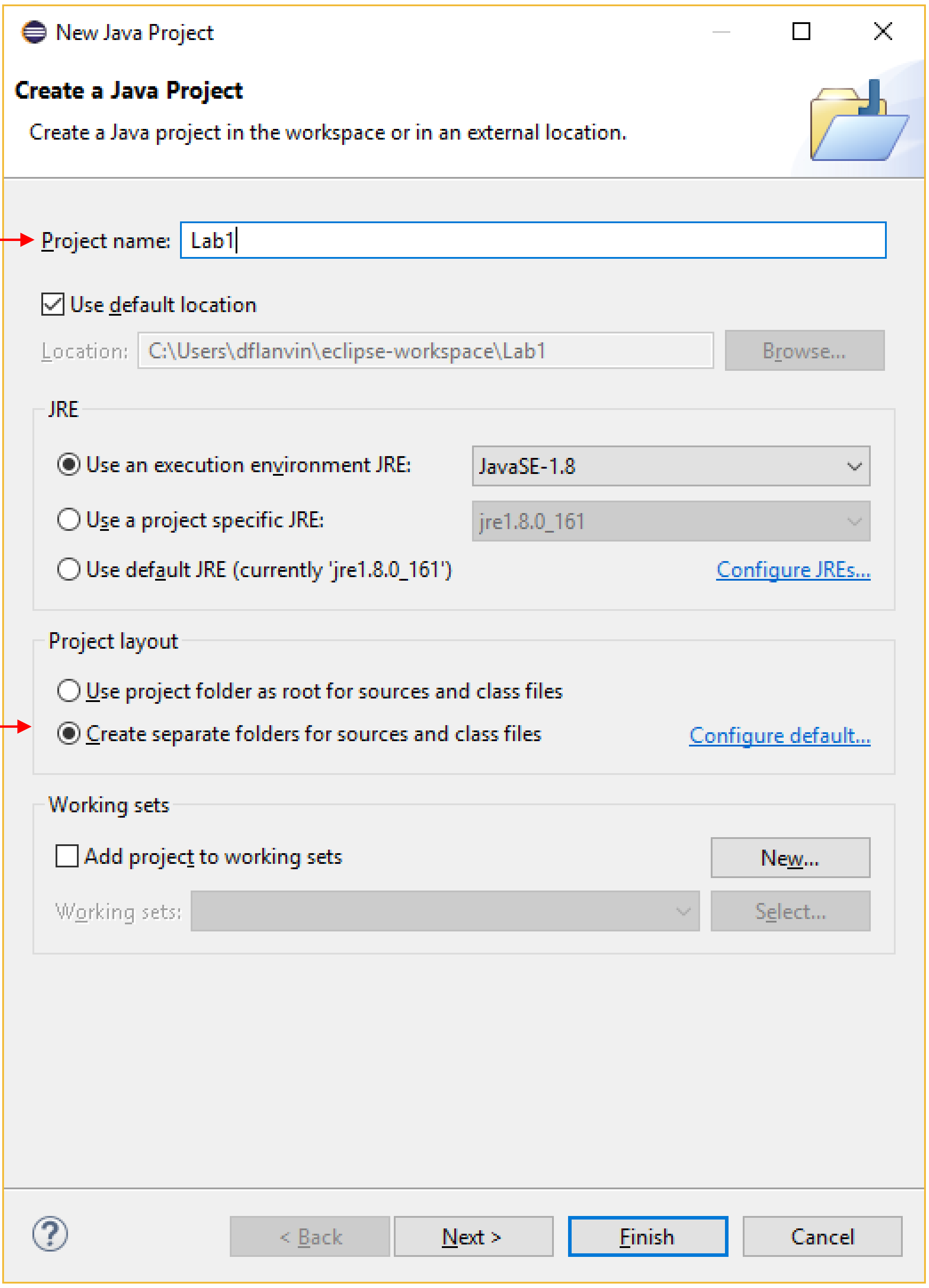
In the following weeks we will discover that this can be a very simple task if we use the appropriate tools and plugins. However, today we will do it manually from the scratch, writing all the Java code we need in order to understand what happens behind the scenes when we "paint" a user interface with Eclipse or any other automatic tool.

2. Creating a project in Eclipse

Let's create a new Java project using *File* | *New* | *Java Project*


File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Java Project
Project...

- **Insert the name of the project**: Write *Lab1* in the field *Project Name*


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Lab1

[✓] Use default location

Location: C:\Users\dflanvin\eclipse-workspace\Lab1

JRE

(●) Use an execution environment JRE: JavaSE-1.8

( ) Use a project specific JRE: jre1.8.0_161

( ) Use default JRE (currently 'jre1.8.0_161')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

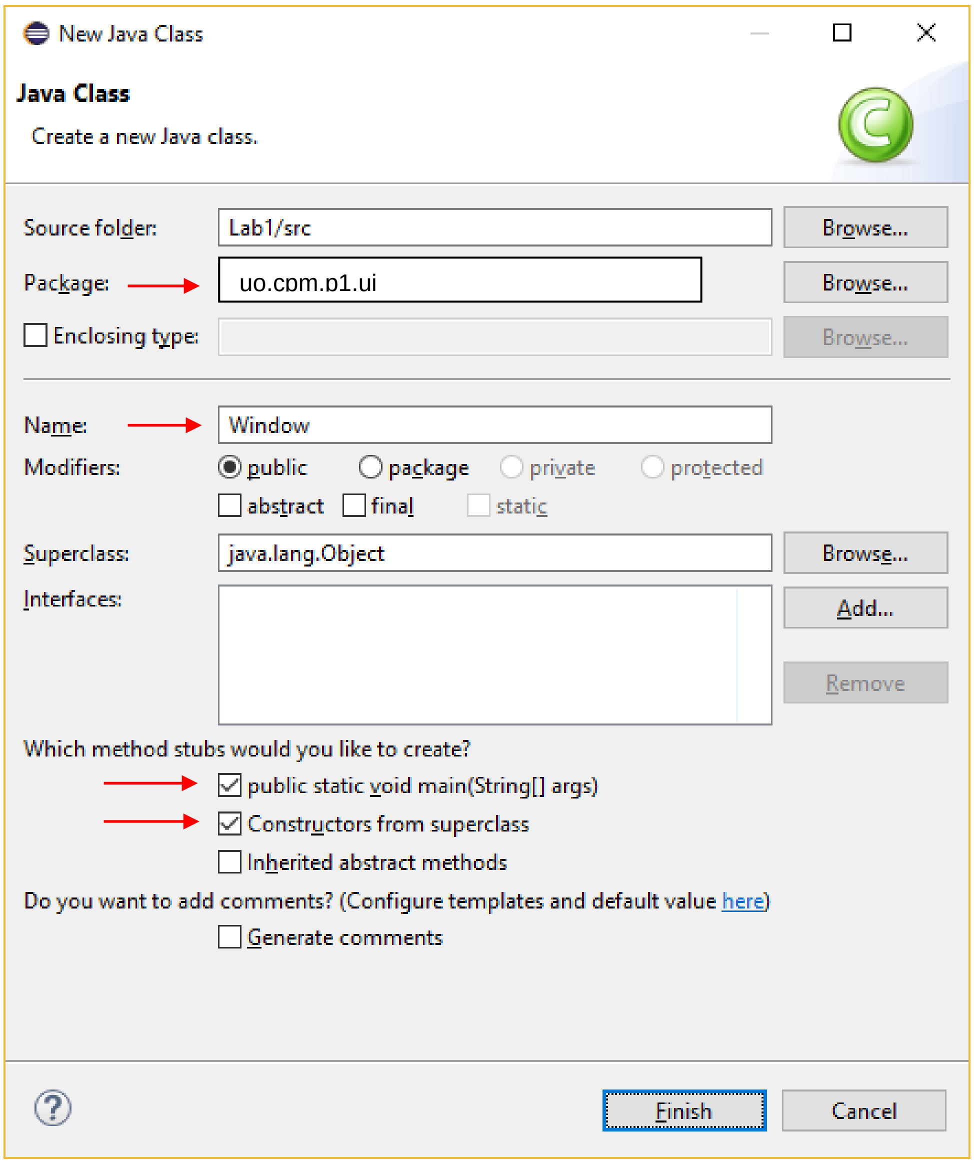
[ ] Add project to working sets

Working sets:

- **Press finish!**

3. Add a new class to the project

This first application will be based in just one Java class that we will call “Window”. We will define the GUI in this class. Start the “new class” creation wizard by File | New | Class. Insert a name for the package, the name of the new class and check the “public static void main” and “constructor from superclass” options before pressing *Finish*.


New Java Class

Java Class

Create a new Java class.

Source folder: Lab1/src
Package: uo.cpm.p1.ui
[ ] Enclosing type:

Name: Window
Modifiers: [✓ public] package private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[✓] public static void main(String[] args)
[✓] Constructors from superclass
[ ] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

The new generated Java code will be as the following:

```java
package uo.cpm.p1.ui;

public class Window {
	public Window() {
		// TODO Auto-generated constructor stub
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
	}
}
```

If we want the class to show a main window, we need to force it to extend the JFrame class. This one belongs to the Swing library.

```
public class Window extends JFrame
```

Given the way operating systems work, a window must be able to be serialized, so we must generate its serial version uid.

```
public class Window extends JFrame {
	private static final long serialVersionUID = 1L;
```

Let's prepare the class. Create an instance of *Window* in the main method, and add a new private method called *initialize* (similar to the way we will proceed in future labs) in which we will implement every initializing sentence of the components of the window. We call this method from the constructor of the class.

```
package uo.cpm.p1.ui;
import javax.swing.JFrame;

public class Window extends JFrame {
	private static final long serialVersionUID = 1L;
	public Window() {
		initialize();
	}

	private void initialize() {
		// TODO Auto-generated method stub

	}

	public static void main(String[] args) {
		Window window = new Window();
	}
}
```

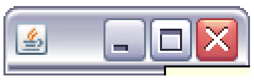
1. Test the application!

Execute the application. Notice that the Windows is not shown in the screen. Why? That's because we must turn it visible modifying the *visible* attribute. We can do that either in the main method or in the initialize method.

```
public static void main(String[] args) {
		Window window = new Window();
		window.setVisible(true);
}
```

}

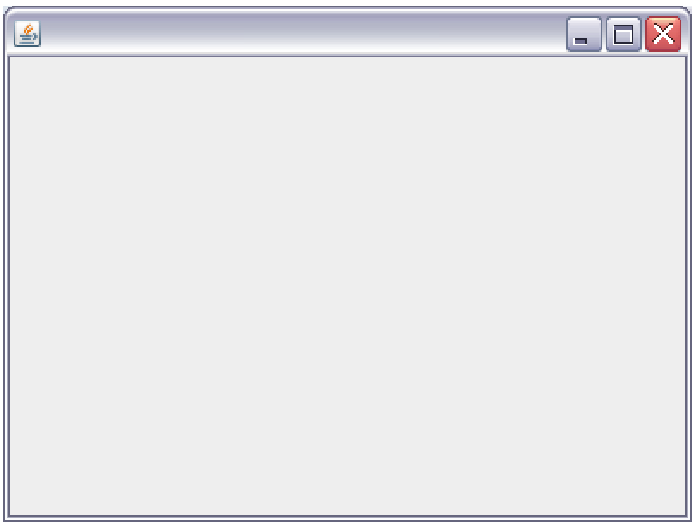
Now, we should see that:



We still need to resize the window from the *initialize* method. We do that using the method *setSize(width,height):*

```
this.setSize(400,300);
```

We execute the application again…



We can add a title to the window adding the following sentence to the initialize method:

```
this.setTitle ("Main Window");
```

2. Adding a container and a button

So far, we already created a window or frame where we will add a container and a button. Every control we add to a window must be contained in a *container* class.
First of all, we will declare two new attributes in the Ventana class, one *JPanel* (the container) and one *JButton* (the button):

```
 public class Window extends JFrame {

	private JPanel contentPane=null;
	private JButton btnOk = null;

	private static final long serialVersionUID = 1L;

	public Window() {
		initialize();
	}
```

We create the main panel and we associate it to the window from the initialize method as follows:

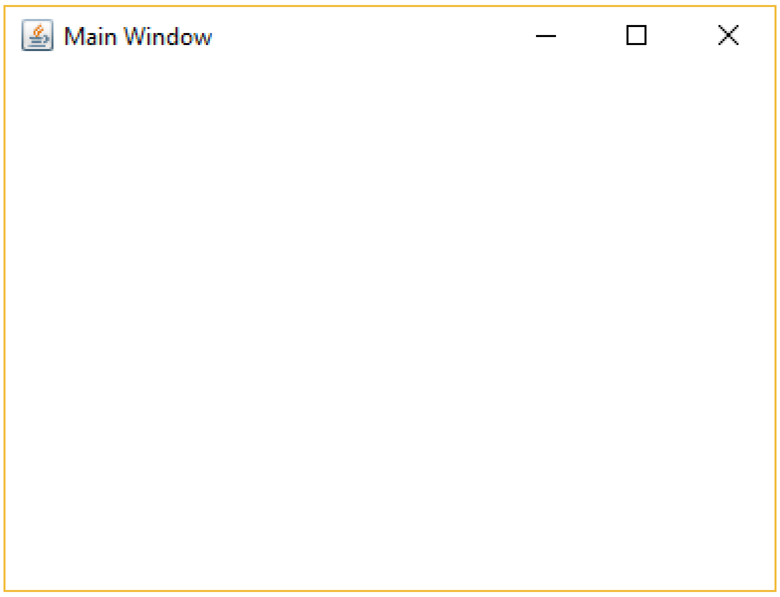
```
contentPane = new JPanel();
this.setContentPane(contentPane);
```

We will also set the background color to white. We will need to import java.awt.Color package.

```
    contentPane.setBackground(Color.WHITE);
```

```
private void initialize(){
    this.setSize(400,300);
    this.setTitle("Ventana Principal");

    panelPrincipal = new JPanel();
    panelPrincipal.setBackground(Color.WHITE);
    this.setContentPane(panelPrincipal);

}
```

Execute!:



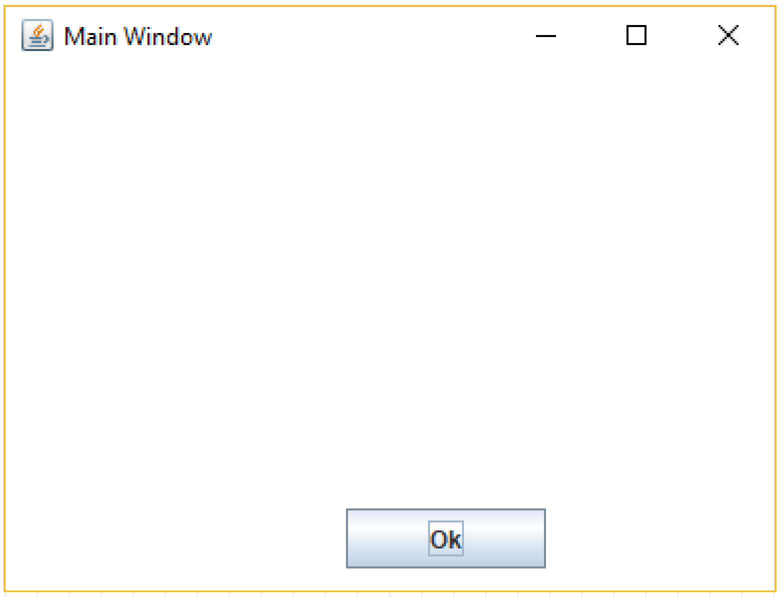
Once we have the wonderful white window, we can proceed with the button. We have to:

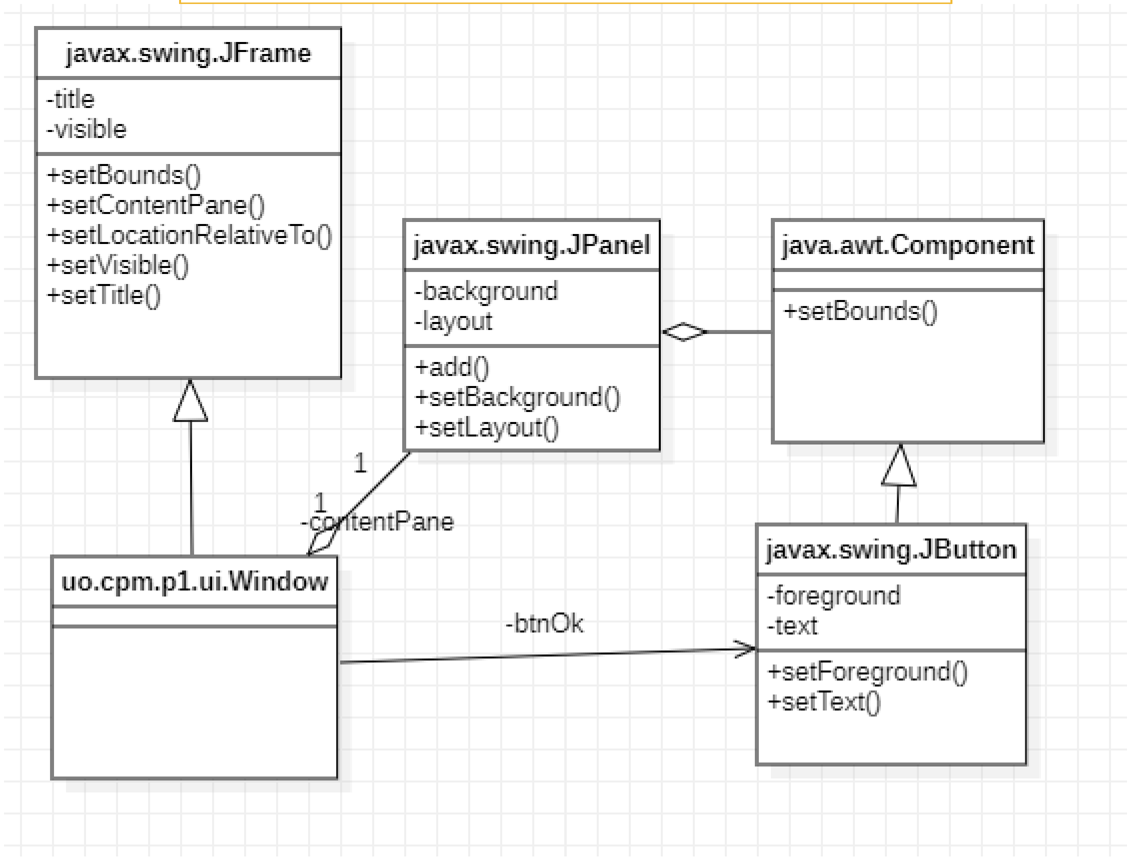
- Create the button and modify its attributes:
    - We create the button inside the *initialize* method
    - We resize the button with *setBounds(x,y,width, height)*
    - We add a text with *setText("Ok");*
- Now, we have to put the button inside the panel:
    - We modify the layout of the panel, setting it *null.*
    - We add the button to the conta170iner (method *add*).

```
private void initialize() {
            this.setSize(400,300);
            this.setTitle("Main Window");

            contentPane = new JPanel();
            this.setContentPane(contentPane);

            contentPane.setBackground(Color.WHITE);
            contentPane.setLayout(null);

            btnOk = new JButton();
```

```
		btnOk.setText("Ok");
		btnOk.setBounds(170, 220, 100, 30);

		contentPane.add(btnOk);
				}
```

Now, we are ready to execute it again:





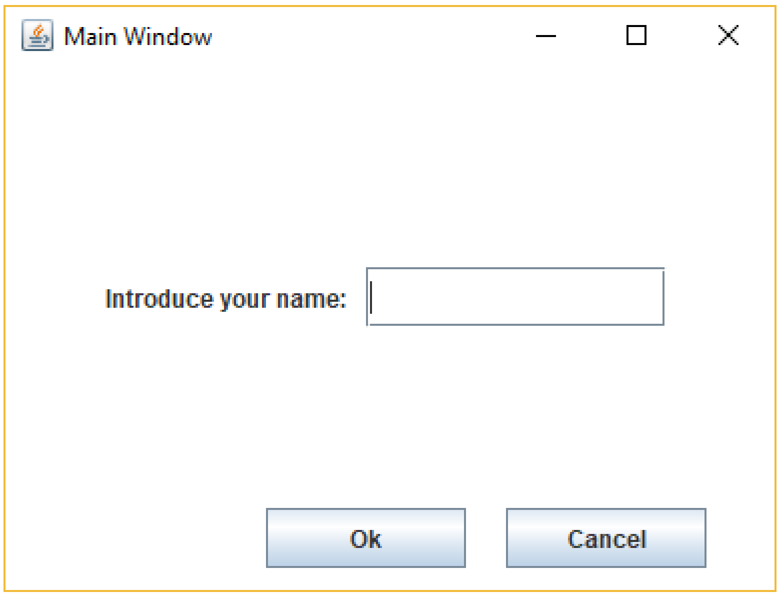
3. Center the windows in the screen

The window is by default on the left-upper corner of the screen. The next code must be added to the main method after creating the window**, but before turning it visible**, in order to put it in the middle of the screen.

```
window.setLocationRelativeTo(null);
```

Tasks Lab 1:

1. **IMPORTANT**: Study the theory foundations associated to this lab. Remember that all these theory pills are part of the theory contents and will be evaluated in both the theory and lab exams.
2. **Document** (with Javadoc comments) the application and the proposed extensions code.
3. In the same application,
   a) Change the color of the text of the button to blue, by means of the *foreground* property.
   b) Add a new *Cancelar* button.
   c) Add a label (class *JLabel*) and a text field (*JTextField*) in order to allow the user to introduce the username.


Main Window
Introduce your name:
Ok
Cancel