CODAPPS

Designing the layout of the app

Clément Levallois

Version 1.0, last modified

Table of Contents

early makers | em lyon business school

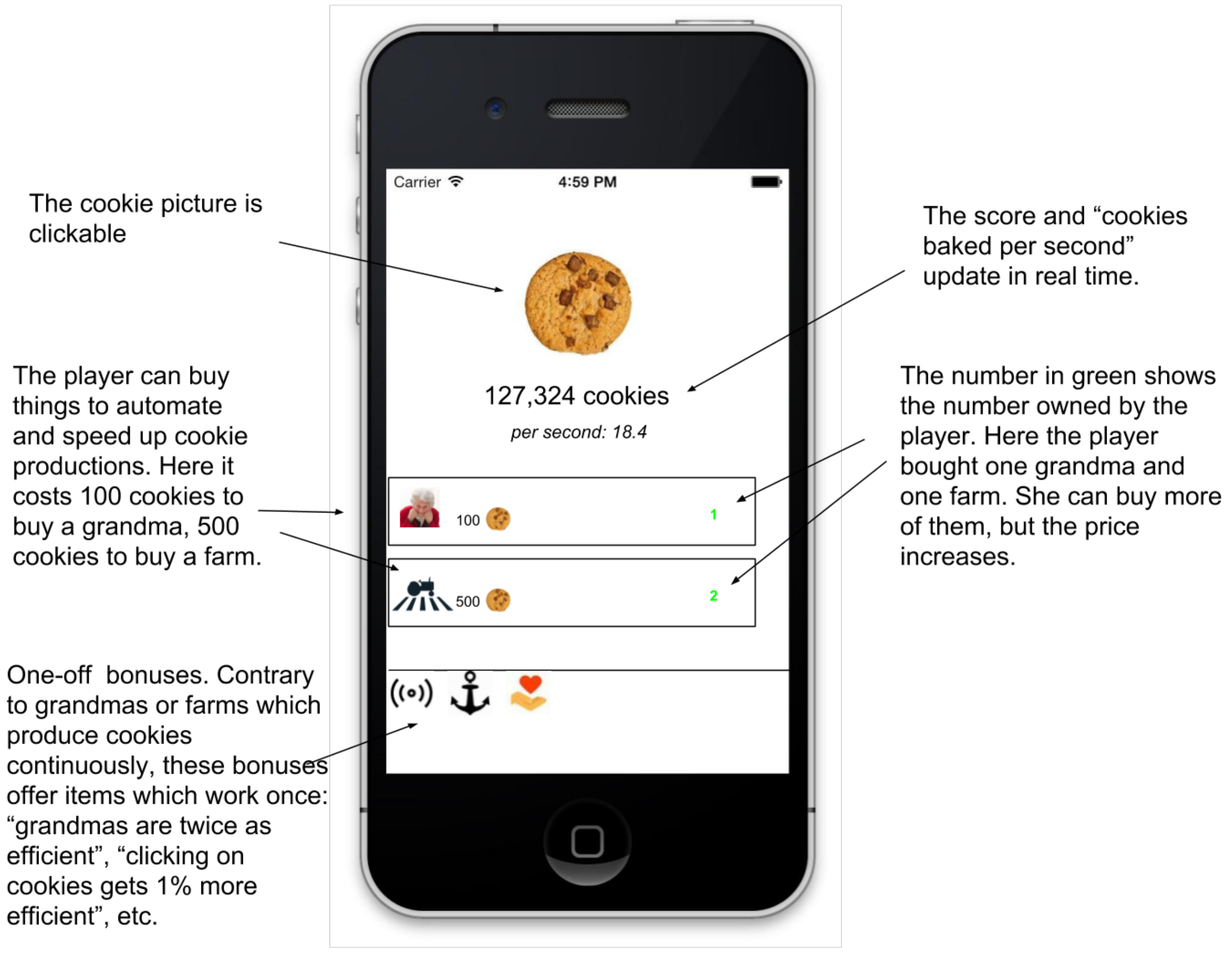
you must have worked and understood the content of Module 5 about **coding** before you can follow this lesson.

1. Create a new project

Take the same steps as in this lesson in Module 1.

1. Creating the main screen of the app

The main action of the game takes place on a single screen:



The cookie picture is clickable

The score and “cookies baked per second” update in real time.

The player can buy things to automate and speed up cookie productions. Here it costs 100 cookies to buy a grandma, 500 cookies to buy a farm.

The number in green shows the number owned by the player. Here the player bought one grandma and one farm. She can buy more of them, but the price increases.

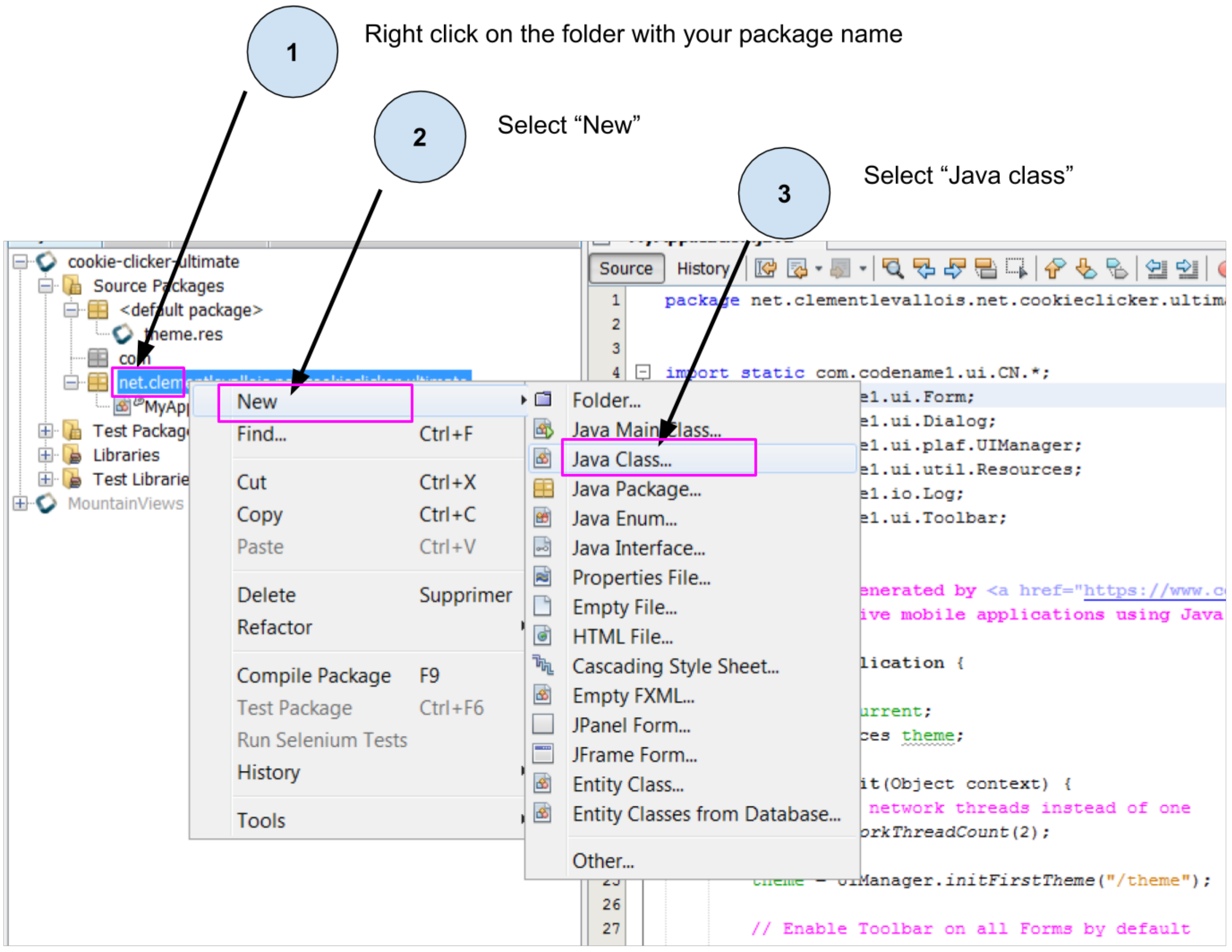
One-off bonuses. Contrary to grandmas or farms which produce cookies continuously, these bonuses offer items which work once: “grandmas are twice as efficient”, “clicking on cookies gets 1% more efficient”, etc.

Figure 1. The design of the cookie clicker mobile app

So the first step is to create this screen. **We will not use the GUI Builder for this** (contrary to previous lessons) as it is too slow and unpractical for a complex app like this one.

a. Create a Form with by coding (not with the GUI Builder!)

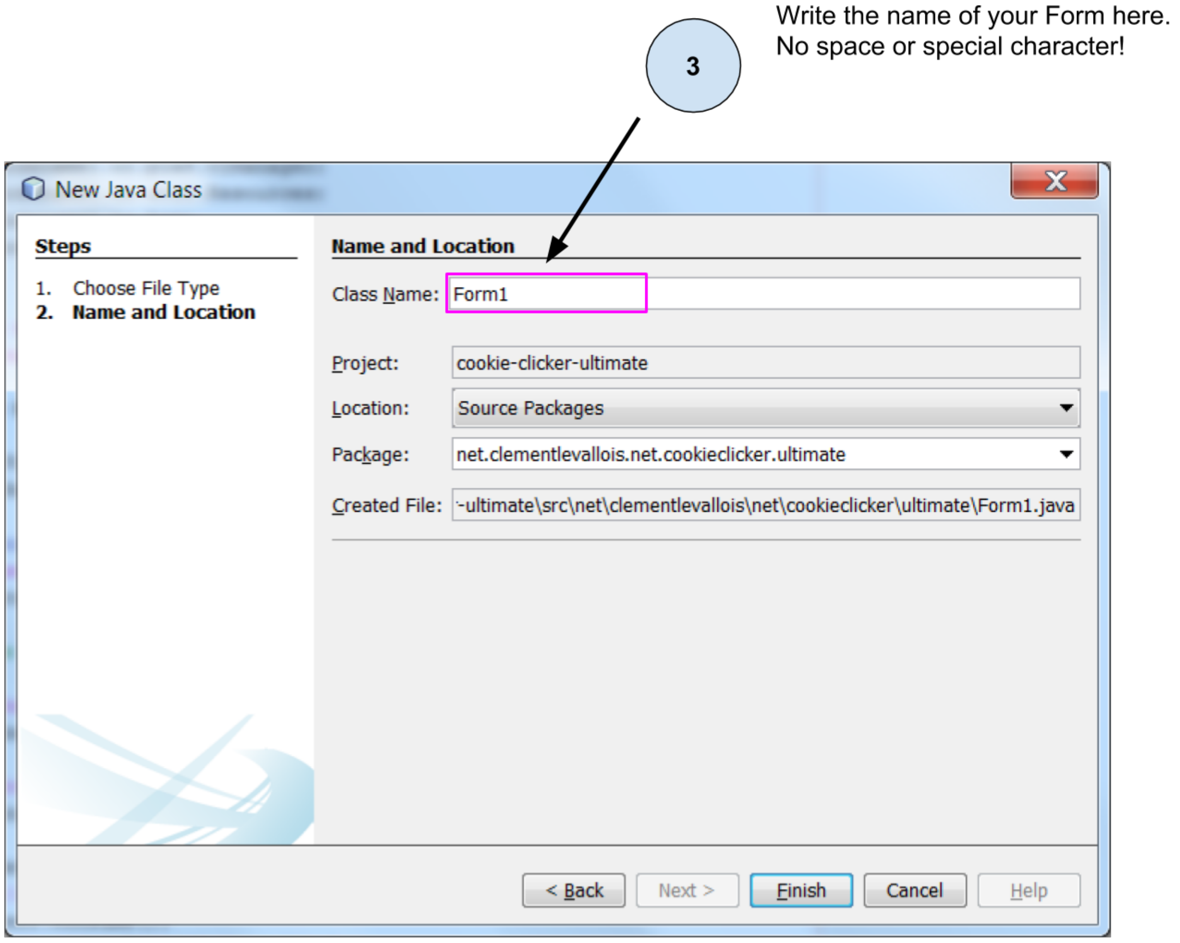
Follow these steps:



```
package net.clementlevallois.net.cookieclicker.ultim

import static com.codename1.ui.CN.*;
e1.ui.Form;
e1.ui.Dialog;
e1.ui.plaf.UIManager;
e1.ui.util.Resources;
e1.io.Log;
e1.ui.Toolbar;

enerated by <a href="https://www.c
ive mobile applications using Java

lication {

urrent;
ces theme;

it(Object context) {
 network threads instead of one
orkThreadCount(2);

theme = UIManager.initFirstTheme("/theme");

    // Enable Toolbar on all Forms by default
```

Figure 2. Creating a Form with code

In the window which opens, **leave every parameter unchanged** except for the name of the screen:



Class Name: Form1

Project: cookie-clicker-ultimate

Location: Source Packages

Package: net.clementlevallois.net.cookieclicker.ultimate

Created File: -ultimate\src\net\clementlevallois\net\cookieclicker\ultimate\Form1.java

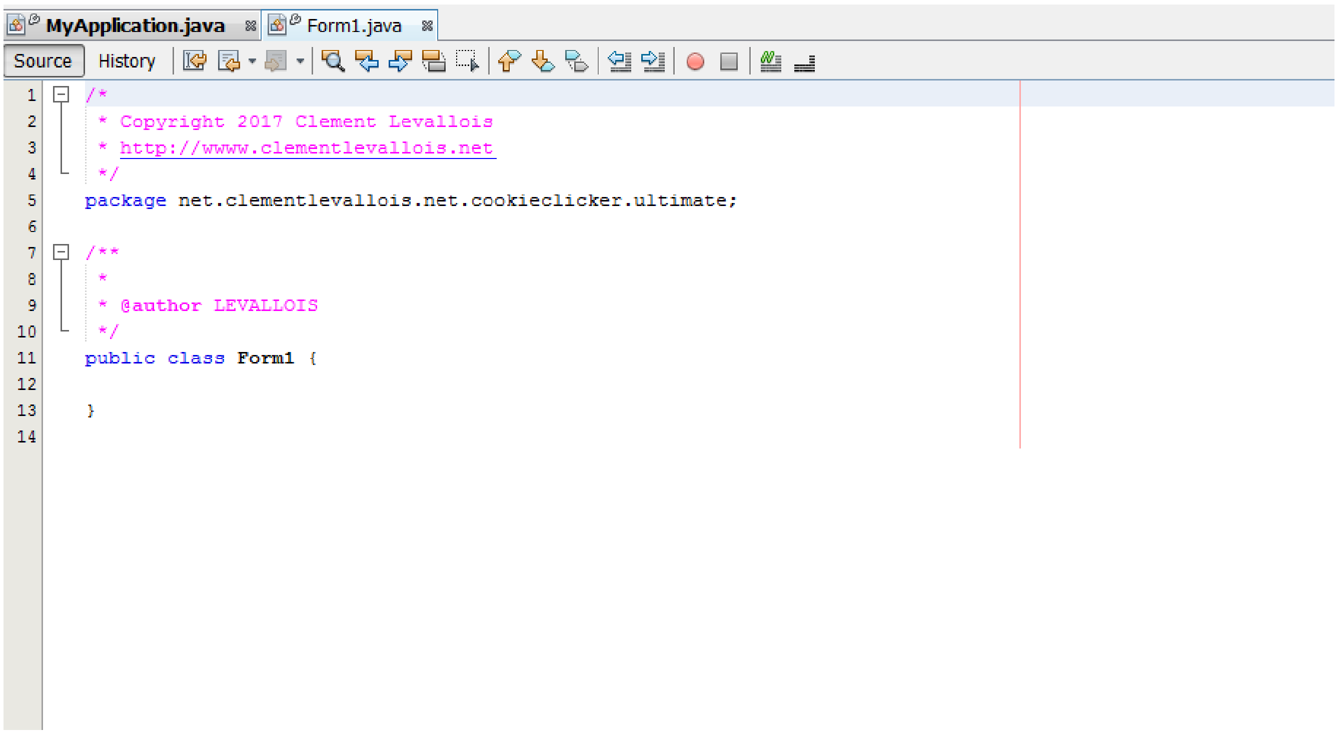
Figure 3. Choosing the name of your Form

This creates a file name `Form1.java` (if you chose Form1 as a name).

If you remember the lesson on `methods and classes` in Module 5, you should realize we just created a `class`.

The role of classes is to organize the code in different files and subunits, for a cleaner style of coding.

At the moment the class is almost empty:



```
/*
 * Copyright 2017 Clement Levallois
 * http://wwww.clementlevallois.net
 */
package net.clementlevallois.net.cookieclicker.ultimate;

/**
 *
 * @author LEVALLOIS
 */
public class Form1 {

}
```

Figure 4. The form has been created and is almost empty

In this class, **write or copy paste** 2 lines of code, so that the entire file looks like:

Creating a function which runs when the Form is instantiated

```
package net.clementlevallois.net.cookieclicker.ultimate; ①

public class Form1 {

    public Form1() {  ②
    } ③

} ③
```

① of course your package name will be different

② the function should have the same name as your class: Form1 in this case.

③ Don't forget that there are two } to close at the end!

What are these two lines we added

```
    public Form1() {  ①

    } ①
```

① anything we write between these 2 curly braces **will be executed when** `Form1` **is instantiated**.

So this is a special method: when the class is instantiated, it will run (just once).

Another way to say it is: "when the objet `Form1` is built, this method defines how it is constructed."

For this reason it is called a `constructor`.

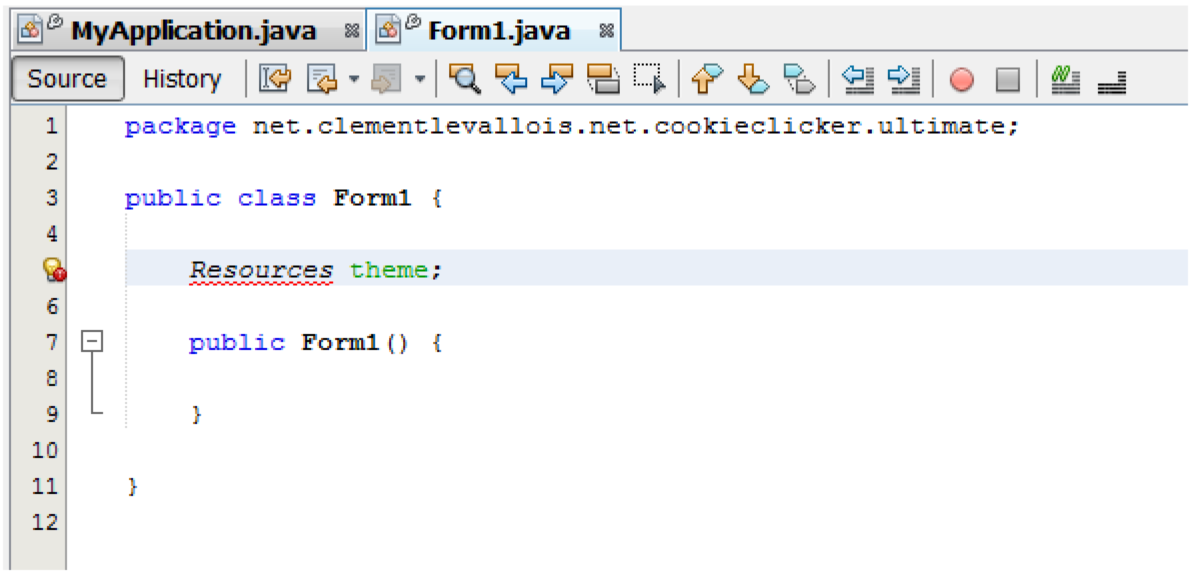
Next, let's add our first variable:

Creating a function which runs when the Form is instantiated

```
package net.clementlevallois.net.cookieclicker.ultimate; ①

public class Form1 {
    Resources theme;  ①

    public Form1() {
    }

}
```

① our first variable! It is an object useful to access the pics we need in the app. We'll use it later

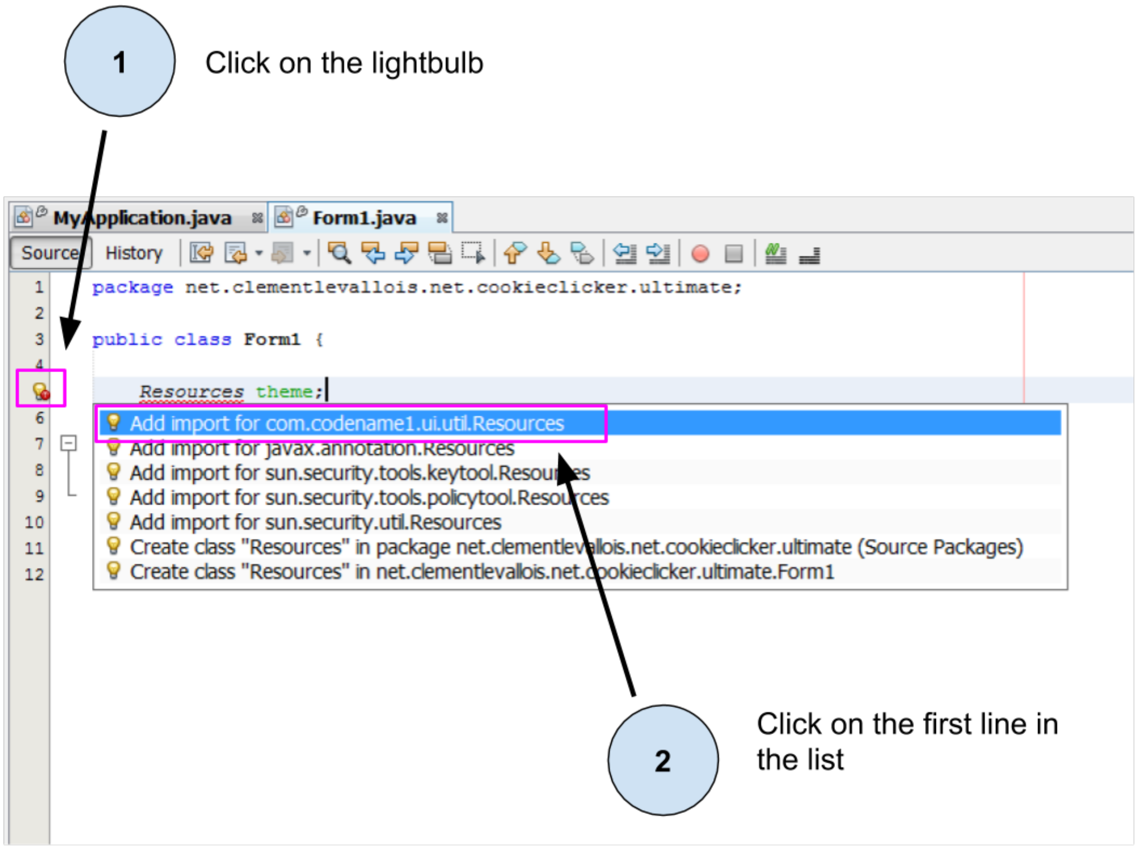
You should have the same error signal as this one :



```
package net.clementlevallois.net.cookieclicker.ultimate;

public class Form1 {

    Resources theme;

    public Form1() {

    }

}
```

Figure 5. Error when creating a variable of type Resources

Follow the steps to fix the error:

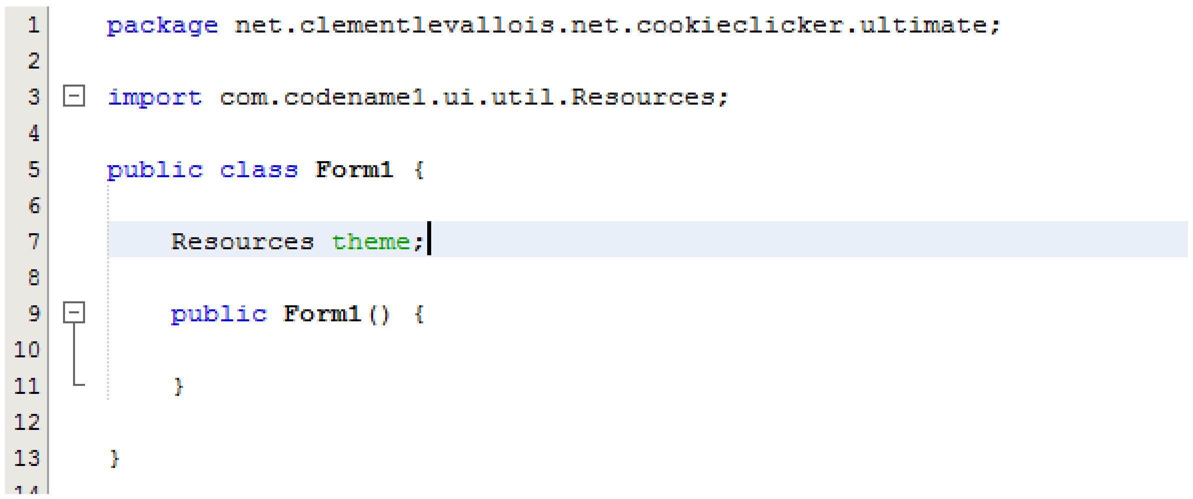


```
package net.clementlevallois.net.cookieclicker.ultimate;

public class Form1 {

    Resources theme;
```

```
Add import for com.codename1.ui.util.Resources
Add import for javax.annotation.Resources
Add import for sun.security.tools.keytool.Resources
Add import for sun.security.tools.policytool.Resources
Add import for sun.security.util.Resources
Create class "Resources" in package net.clementlevallois.net.cookieclicker.ultimate (Source Packages)
Create class "Resources" in net.clementlevallois.net.cookieclicker.ultimate.Form1
```

Figure 6. Importing the type

The result is a new line added on top: see line 3 here:



```
1   package net.clementlevallois.net.cookieclicker.ultimate;
2
3   import com.codename1.ui.util.Resources;
4
5   public class Form1 {
6
7       Resources theme;
8
9       public Form1() {
10
11      }
12
13  }
```

Figure 7. a line telling where the type is coming from

What you did is simple:

- you created a variable of type `Resources`, but this type is not included in the basic package of Java (which just includes `String`, `Integer`, etc.)
- We must tell Java **where to find it so that it can import it**:

- we chose `import com.codename1.ui.util.Resources` because the type `Resources` comes from Codename One, indeed.

Finally, let’s add a last element:

Adding an info that our class extends another class

```
package net.clementlevallois.net.cookieclicker.ultimate;

import com.codename1.ui.util.Resources;

public class Form1 extends com.codename1.ui.Form {  ①
    Resources theme;

    public Form1() {
    }

}
```

① we added `extends com.codename1.ui.Form` before the `{`

What does it mean? We just said that the class we created (`Form1`) **extends** another class, named `com.codename1.ui.Form`

→ our class `Form1` just inherited of all the methods of the class `Form`! → **inherited** means that whatever a `Form` is capable of doing (having Labels in it, be in a BorderLayout or GridLayout, have a title, a background color, etc.), **our class `Form1` inherited these powers and so, it can do it as well!**

With all this setup done, we can work on the layout we want for our screen.

b. Creating the layout of the Form

I mentioned that anything written here would be executed when the Form is instantiated:

The constructor of the class Form1

```
    public Form1() {  ①

    } ①
```

① this method will be executed when `Form1` is instantiated

So we are going to write our layouts there, so that they get into place right when the Form gets created.

I suggest we organize the screen in three big regions:

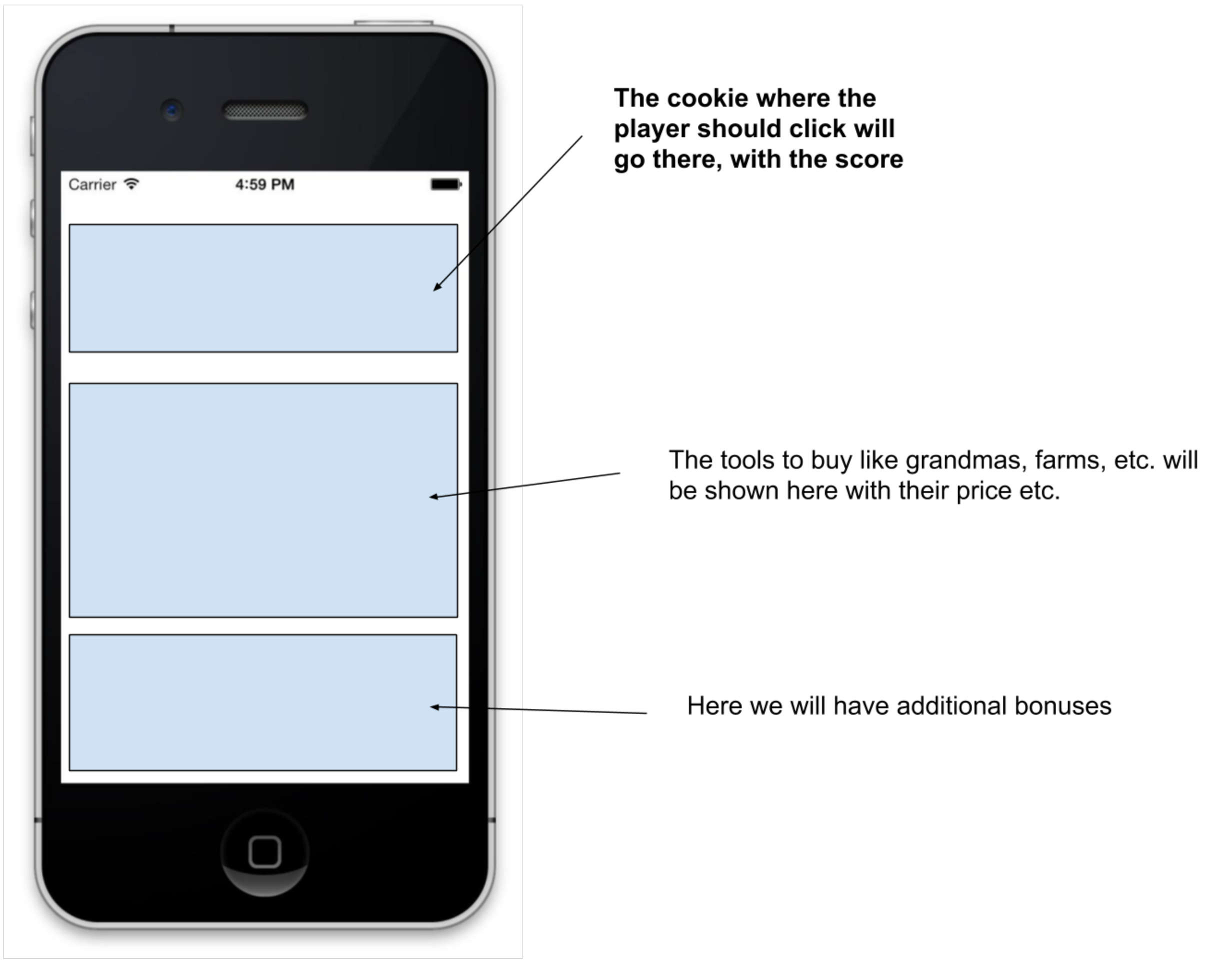

Carrier
4:59 PM
The cookie where the player should click will go there, with the score
The tools to buy like grandmas, farms, etc. will be shown here with their price etc.
Here we will have additional bonuses

Figure 8. The organization of the screen in 3 regions

Putting the Form in a BorderLayout (see the lesson on the BorderLayout) will help us divide the screen in these three regions. GridLayout might be another option.

The idea is to have something like this:

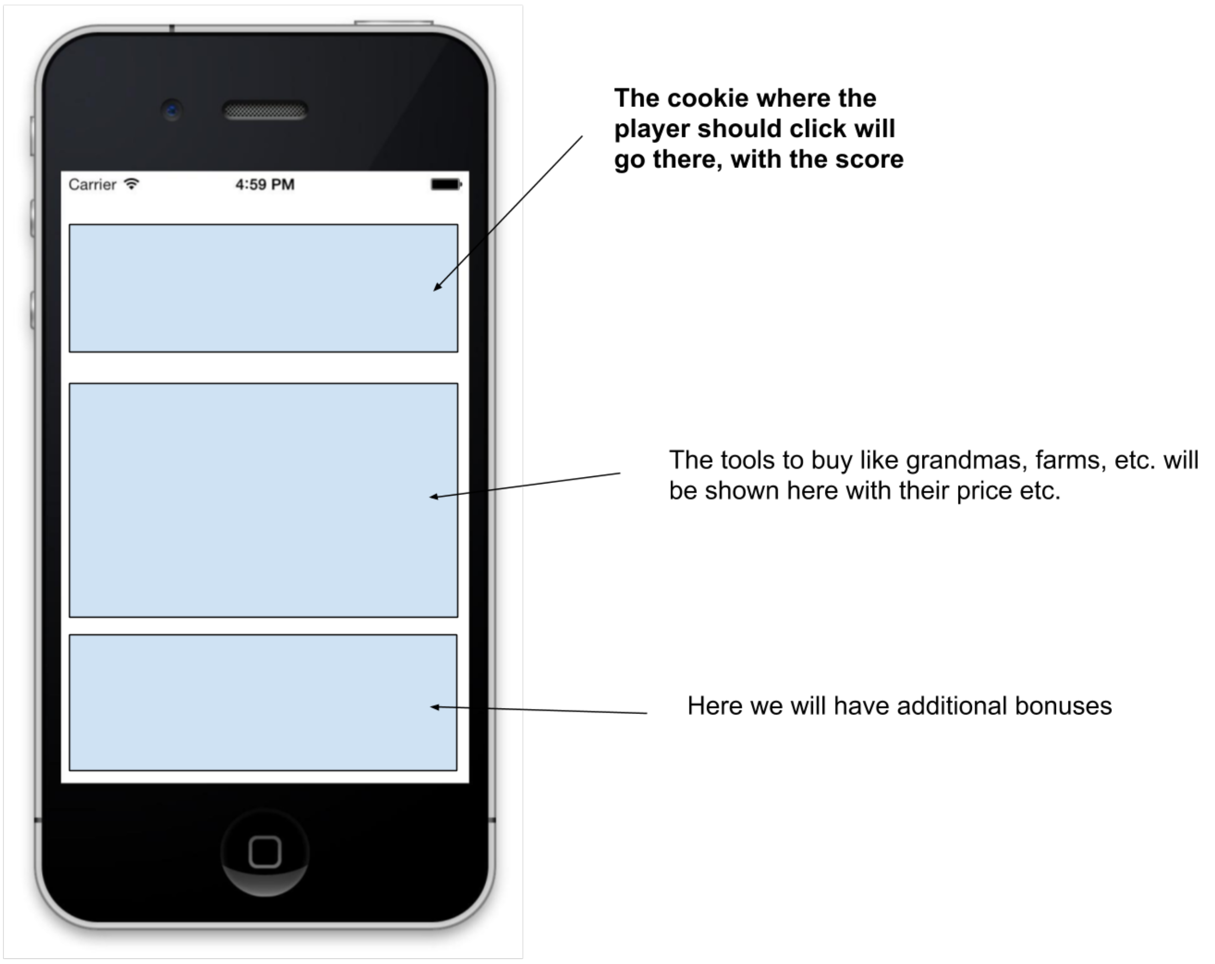

Carrier
4:59 PM
The cookie where the player should click will go there, with the score
The tools to buy like grandmas, farms, etc. will be shown here with their price etc.
Here we will have additional bonuses

Figure 9. The organization of the screen in 3 regions

Let’s define all of this for our Form, with some code:

Start by creating layouts

```
// these lines of code should go inside  public Form1() { }

//we create the different layouts we will need in this Form:
BorderLayout borderLayout = new BorderLayout();
BoxLayout boxYLayout = BoxLayout.y(); ①
BoxLayout boxXLayout = BoxLayout.x(); ①
```

① weird looking code: what is `BoxLayout.y()` and `BoxLayout.x()`? These are static methods, used in a fancy way to do the equivalent of `new BoxLayout()`

Choose a BorderLayout for the Form

```
// these lines of code should go inside  public Form1() { }

this.setLayout(borderLayout); ①
```

① the Form is the file where this code is being written, so we can name it with the keyword `this`.

The Form has a method setLayout which we can use to switch the Form to a BorderLayout.

We now have a Form, set in a BorderLayout. We must put many things in each of the North, Center and South region of the layout.

To organize things, we'll add a container for each region, and our Components will go inside these containers:

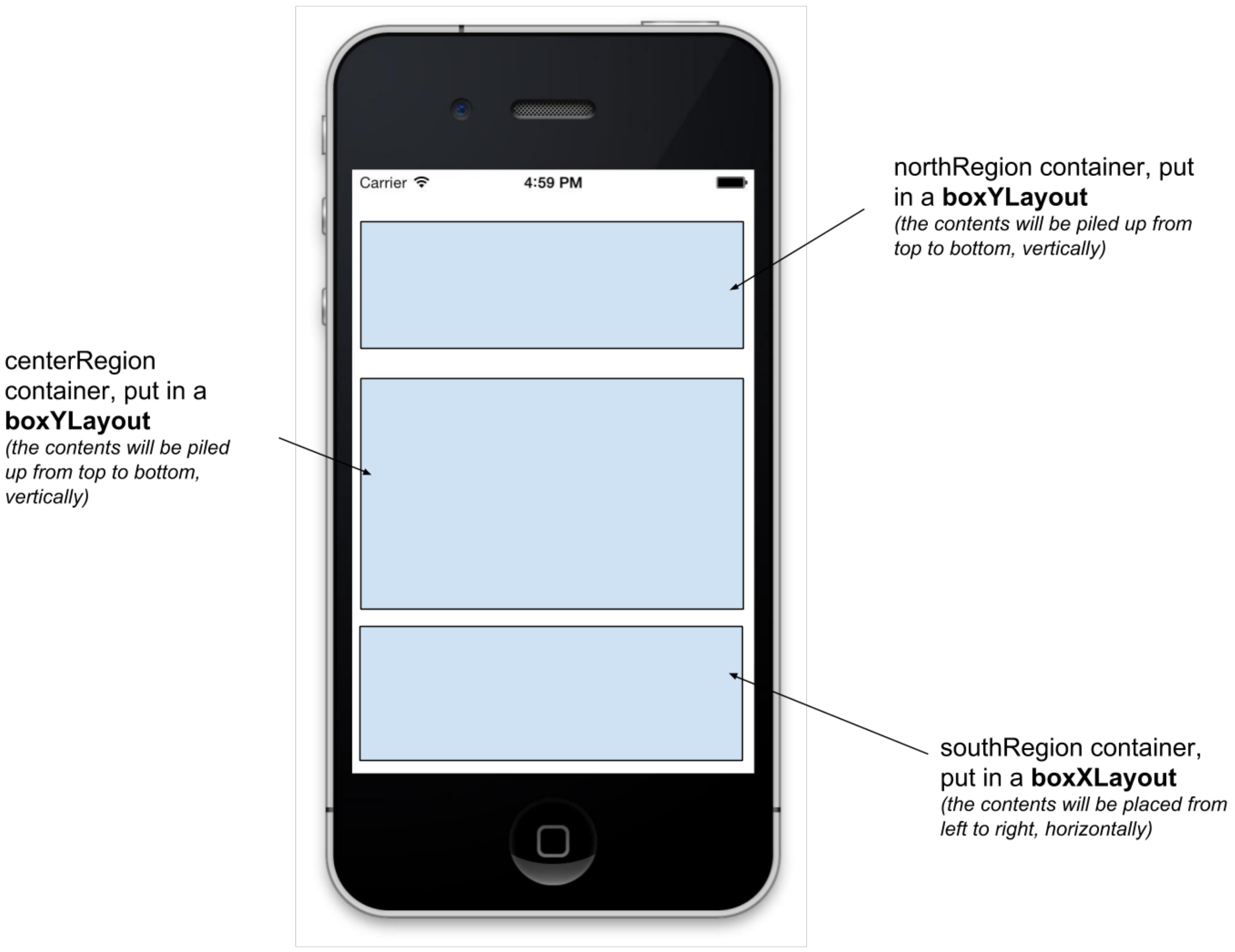

Carrier
4:59 PM
northRegion container, put in a **boxYLayout**
(the contents will be piled up from top to bottom, vertically)
centerRegion container, put in a **boxYLayout**
(the contents will be piled up from top to bottom, vertically)
southRegion container, put in a **boxXLayout**
(the contents will be placed from left to right, horizontally)

Figure 10. Adding 3 containers - one for each region of the Form

[[adding-container-north] .Adding a container to the North region of the Form

```
// these lines of code should go inside  public Form1() { }

//we create a Container that will contain everything in the "North" part of the
BorderLayout:
Container northRegion = new Container();

//we choose a BoxY Layout for this container:
northRegion.setLayout(boxYLayout);
```

Adding a container to the Center region of the Form

```
// these lines of code should go inside  public Form1() { }

//we create a Container that will contain everything in the "Center" part of the
BorderLayout:
Container centerRegion = new Container();

//we put this center region in a Box X Layout:
centerRegion.setLayout(boxXLayout);
```

Adding a container to the South region of the Form

```
// these lines of code should go inside  public Form1() { }

//we create a Container that will contain everything in the "South" part of the
BorderLayout:
Container southRegion = new Container();

//we put this container in the south region in a BoxX Layout:
southRegion.setLayout(boxXLayout);
```

We created several containers, **but it doesn't mean they are in our Form yet**. We need to add them, and tell where they should go:

Adding the containers to the Form

```
// these lines of code should go inside  public Form1() { }
this.addComponent(BorderLayout.NORTH, northRegion);
this.addComponent(BorderLayout.SOUTH, southRegion);
this.addComponent(BorderLayout.CENTER, centerRegion);
```

The end

Questions? Want to open a discussion on this lesson? Visit the forum here (need a free Github account).

Find references for this lesson, and other lessons, here.

This course is designed by Clement Levallois.

Discover my other courses in data / tech for business: http://www.clementlevallois.net

Or get in touch via Twitter: @seinecle