



aicas GmbH

First Experiences with Xlets — User Manual

Version 2.7.1dev
28 October 2016

Released under NDA

aicas

Every effort has been made to ensure that all statements and information contained in this document are accurate. However, aicas GmbH accepts no liability for any error or omission therein.

© Copyright of this document is owned by aicas GmbH, Karlsruhe

Contents

1	JamaicaCAR	3
1.1	Features and Functions	4
1.2	JamaicaCAR Tools	5
1.2.1	JamaicaCAR Emulator	5
1.2.2	CodenameOne Designer	5
1.2.3	Eclipse Plugin	6
2	Xlets	7
2.1	What are Xlets	7
2.1.1	Functions and Features	8
2.1.2	Graphical and Service Xlets	9
3	Creating a Downloadable Application with Eclipse	11
3.1	The javax.microedition.Xlet Interface	11
3.2	Initializing an Xlet Project in Eclipse without the Codenameone Designer	12
3.3	Example: HelloWorld	16
4	Creating a Downloadable Application with the CodenameOne Designer	19
4.1	Initializing the Designer in Eclipse	19
4.2	Designer basics	21
4.2.1	Action Listeners	22
4.2.2	Action Event	23
4.2.3	Containers	24
4.2.4	Layouts	25
4.2.5	Themes	28
A	Further Reading	31

Preface

This manual is aimed at any and all who want/need to work with Xlets in an Eclipse IDE for the first time and require an introduction to this field. Virtually no understanding of Eclipse and only very rudimentary knowledge of Java is required to understand and implement what is said in this manual. This manual is intended as a quick introduction to kick-start the user's first steps with Xlets.

This is a very basic guide to the application and the technicalities of using Xlets in Eclipse and the CodenameOne Designer. This manual also includes a basic introduction to the JamaicaCAR tools used and an illustration of the concept of Xlets. This is intended as a hands-on-guide, therefore it is recommended to actively follow the steps illustrated on your computer.

After having read this manual the user should be able to understand the structure of an Xlet application, make basic Xlet applications using both Eclipse and the CodenameOne Designer according to their level of understanding of Java and easily continue exploring the capabilities of Xlets and the CodenameOne Designer without technical difficulties.

Chapter 1

JamaicaCAR

JamaicaCAR is an automotive application framework. It is based on the JamaicaVM and Java technology, and adds realtime and reliability to the flexibility and extendibility of mobile apps. JamaicaCAR supports 2D and 3D applications using a lightweight user interface toolkit (Codename-One) and OpenGL ES (Graphic Library for embedded systems). Extensive libraries provide access to the board net, internet, GPS information and many other sources. Applications can communicate directly and securely.

The JamaicaVM security model is based on Java's security managers and is extended by signable apps, privileges, and access control mechanisms. The system is controlled by the JamaicaCAR manager, which can pause, stop, or slow down running apps at any time.

OEMs (original equipment manufacturer) can ensure their look and feel for all apps. Suppliers can gain access to the center console and can easily integrate their after market products, if this is allowed by the OEM. For Automobile owners the connection of GPS information and internet access opens a host of new possibilities for local services such as finding the nearest available parking place or hotel room, or even communication between vehicles for fleet management.

Did you know?

JamaicaCAR is also sometimes called AMS, after its core component, the Application Management System.

1.1 Features and Functions

The features of the JamaicaCAR include:

Platform independence

The component based design and platform neutrality of JamaicaCAR enables OEMs to deploy applications on any CPU and OS with a custom look and feel.

Safety and Security

Application security management provides fine grain access control to platform services and limit on resource use by applications.

Compactness and Swiftness

The JamaicaVM, that the JamaicaCAR uses, is equipped with a highly optimizing static compiler and profiler. The tradeoff between runtime performance and code size can be chosen freely.

By using precompiled JARs the runtime performance is further increased.

Dynamic Loading

The JamaicaVM enables dynamic class loading for upgrading applications at runtime.

Remote Debugging

Applications running on a target system can be debugged with standard IDEs such as Eclipse and NetBeans.

HMI (Human Machine Interface)

Both 2D and 3D graphics support, using a Light Weight User Interface Toolkit and OpenGL ES. This, with hardware acceleration, provides ease of development and maximum performance.

1.2 JamaicaCAR Tools

1.2.1 JamaicaCAR Emulator

The JamaicaCAR Emulator (see Figure 1.1) facilitates the testing of applications greatly. By using the JamaicaCAR Emulator, users can bypass the need to install a written application on an external device in order to find errors and bugs in the code. With the Emulator you can test your applications from the comfort of your very own desktop. The Emulator, that was developed by aicas GmbH, emulates the interface of an external device on your computer's desktop directly from your IDE (Integrated Development Environment).

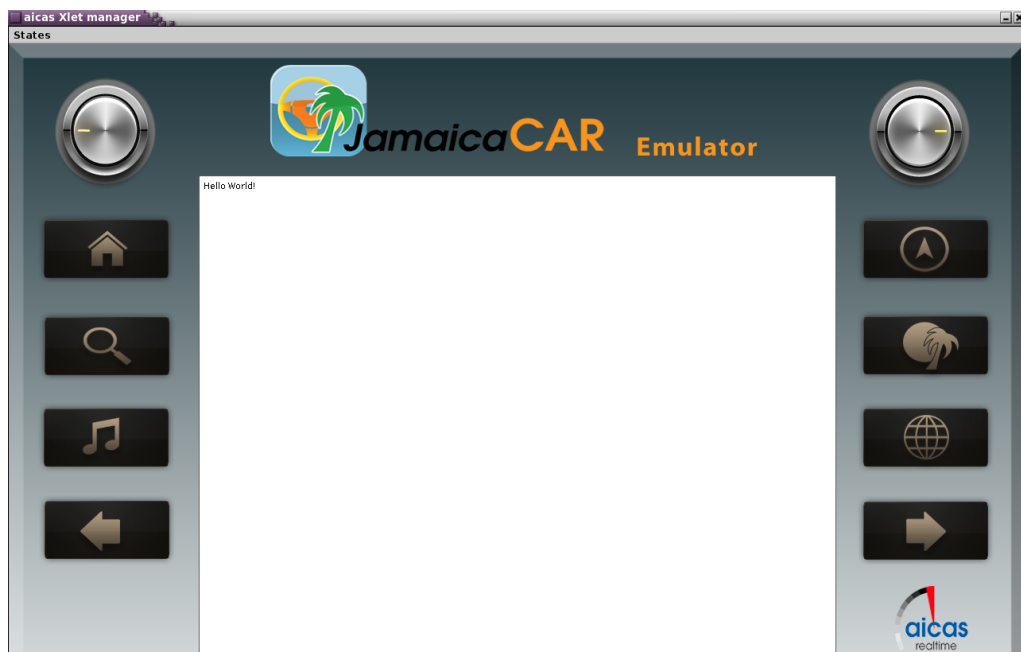


Figure 1.1: The JamaicaCAR Emulator

1.2.2 CodenameOne Designer

Codename One is a set of tools for mobile application development that derive a great deal of its architecture from Java. It stands both as the name of the startup that created the set of tools and as a prefix to the distinct tools that make up the Codename One product. The goal of the Codename One project is to take the complex and fragmented task of mobile device programming and unify it under a single set of tools, APIs and services to create a more manageable approach to mobile application development without sacrificing development power/control.

—<http://www.codenameone.com/developer-guide.html>

The CodenameOne Designer (see Figure 1.2) is one of the set of tools provided by CodenameOne. It is a GUI Builder which simplifies the production of a GUI immensely. The simple point and drag

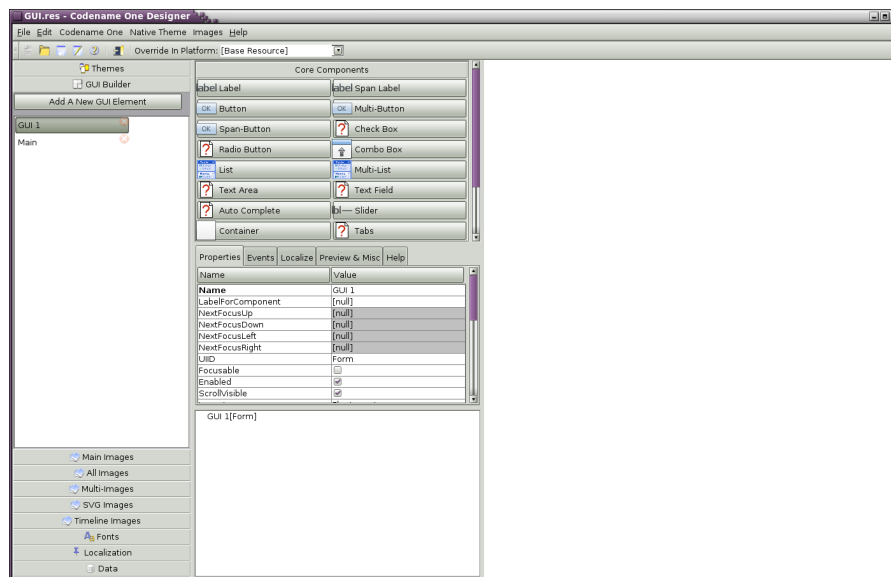


Figure 1.2: The CodenameOne Designer

interface is easy to operate and doesn't require any code to be entered in order to create a tremendous variety of themes, styles, components and visually pleasing surfaces.

The biggest advantage of the CodenameOne Designer is that it is written entirely in Java, it draws its own interface and handles its own events/states. This has huge portability advantages since the same code executes on all platforms. The GUI builder allows for live preview and accurate reproduction across platforms.

Once constructed using the Designer, an application can be further developed in the AMS without further ado and can be downloaded on to a device directly.

1.2.3 Eclipse Plugin

The JamaicaVM Eclipse Plugin is another product of aicas GmbH. Without the Eclipse Plugin none of the above tools would work together with the Eclipse IDE. The Eclipse Plugin is also compatible with most Eclipse derivative IDEs like Windriver's Workbench or QNX's Momentics. All of the JamaicaVM tools can be run from command line, output to stdout (standard output) and can be integrated with any IDE that supports launching external tools.

We are very excited to be working with aicas to create this world-class mixed language technology, JamaicaVM provides the industry's most advanced real-time Java technology. Soon, developers will be able to use this technology in a mixed language environment combining real-time Java with C, Embedded C++, and Ada. This unique environment should be very attractive not only to Java developers who require mixed language capability, but also to legacy developers who are looking to migrate existing C, Embedded C++ and Ada programs to Java.

–Bob Morris, president and CEO of DDC-I.

Chapter 2

Xlets

2.1 What are Xlets

Xlets are a base class for applications that run on the JamaicaCAR. Applications written with the JamaicaCAR framework are always written using Xlets. Xlets comprise very small programs, written in Java; in this they are very similar to Java applets.

A Java applet is a special kind of Java program that a browser enabled with Java technology can download from the internet and run. An applet is typically embedded inside a web page and runs in the context of a browser. The Applet class provides the standard interface between the applet and the browser environment.

–docs.oracle.com/javase/tutorial/deploymentapplet

Did you know?

Xlets were first used in Sun's Java TV specification to support Digital TV.

An Xlet application consists of multiple Xlet states that work together. They are the building blocks of the application. The user writes his program in the Xlet methods. Xlet methods tell the application what to do in each Xlet state and when to switch between states. There are four main Xlet methods to an application which perform the following transitions:

- **initXlet**
This Xlet method is the first one to activate when the application is started. 'init' is short for 'initialize' and that is exactly what it does. This Xlet typically only runs once in an application. The display is initialized here, the code for this is automatically generated when you start a new Xlet class. initXlet sends the application into the stable 'Paused' state.
- **startXlet**
The startXlet method starts the stable 'Running' state and tells the Application what to do while in the Running state. This method can be called from either the Paused or the Running state. While the application is in this state, it is running in the foreground. This is where some applications do what they are meant to do. (I say 'some' applications because we distinguish between graphical and service Xlets, more on these later). Here structures are made visible for the user on the GUI (Graphical User Interface) and the user can interact with the application insofar as it was programmed.

- `pauseXlet`
This method sends the Xlet into the 'Paused' state and determines what is done in the Paused state. It can do this either from the 'Running' state or from the 'Paused' state, restarting and repeating the state. When in the 'Paused' state, the application relinquishes its control over the GUI and runs in the background. Effectively it pauses any progress it made in the Running state. Unless of course the programmer wrote some code into the method. In this case the Paused state is a service Xlet.
- `destroyXlet`
This is the final method of the application and can be called from any state. Everything is 'cleaned up', i.e. variables are forgotten, memory space that was used in the course of the application is freed up. The Application is terminated.

2.1.1 Functions and Features

If built with Xlets, an application gains the following advantages:

- More than one application can be active at any given time. Though only one application can run in the Running state, visible on the device. However any number of other applications can run in the Paused state at the same time. As a result of this, the advantage of not having to restart an application again after switching to a different one is gained.
- Since the code is effectively separated into multiple parts, the layout is considered 'asynchronous'. This allows seamless switching between states and switching to the desired part of code becomes much easier and faster.
- How the different states are related and where which methods are able to be called, is depicted in Figure 2.1.

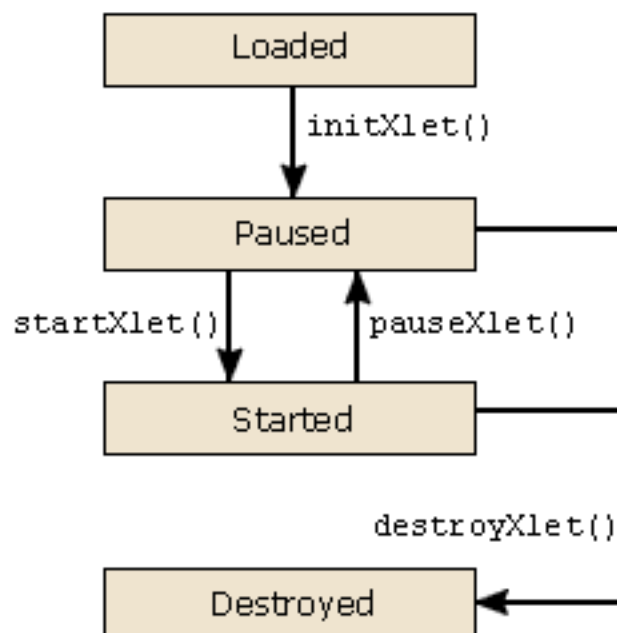


Figure 2.1: Xlet states

2.1.2 Graphical and Service Xlets

As promised, we will now take a quick look at graphical and service Xlets. They are really very simple:

Graphical Xlets

Graphical Xlets are the Xlets that have been focused on so far. A graphical Xlet gives output. It uses the GUI to display, for example, a form with which the user can interact that contains buttons or textfields or a map. It is the typical application that you use in your everyday life when you use your phone to select a contact, play music, shedule an appointment, play a game or any number of other things that you can *see*.

Service Xlets

Service Xlets on the other hand, are essentially all the applications that are not 'graphical'. These are applications that run in the background (in the pauseXlet state) and provide (or 'serve' as in 'service' Xlet) the graphical Xlets with all sorts of information. They do not have any output of their own. Often they are used to enable interaction and communication between different applications. As such, they are also referred to as *Daemon Xlets*.

Daemons are processes that live for a long time. They are often started when the system is bootstrapped and terminate only when the system is shut down. Because they don't have a controlling terminal, it is said that they run in the background. UNIX systems have numerous daemons that perform day-to-day activities. They are very lightweight and don't contribute any output to a shell.

Xlets are available as part of the `javax.microedition.xlet` package (see Section 3.1)

Chapter 3

Creating a Downloadable Application with Eclipse

3.1 The javax.microedition.Xlet Interface

In order to create an Application using Xlets, the javax.microedition.Xlet interfaces are required. For more information about the package go to <http://docs.oracle.com/javame/config/cdc/ref-impl/pbp1.1.2/jsr217/javax/microedition/xlet/package-summary.html>

To purchase the package (if you have not already), contact aicas GmbH or aicas incorporated depending on your location. For contact information go to <https://www.aicas.com/cms/en/contact-us> Once you have created an Xlet framework according to the steps below, the imported frameworks should be visible at the top of the code (see Figure 3.1).



Figure 3.1: microedition.Xlet Package content

3.2 Initializing an Xlet Project in Eclipse without the Codename-one Designer

Once you have purchased and downloaded the package, you can start building Xlets using Eclipse.

1. First a new project must be created in Eclipse. The option >New>Other (see Figure 3.2) is used, creating a new Java project will not work.

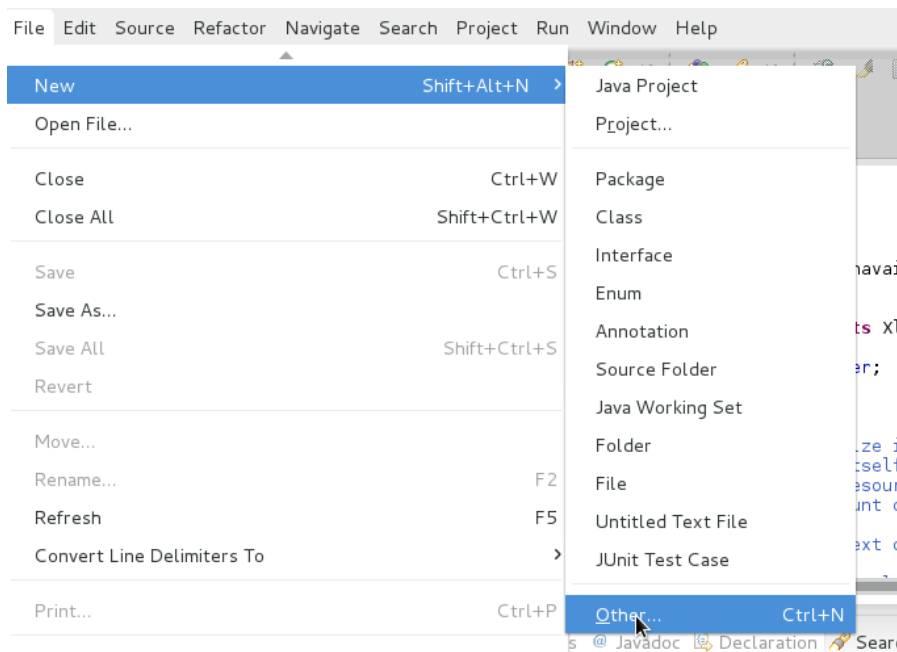


Figure 3.2: Creating an Xlet project in Eclipse

2. Next, from the list of possible projects, choose the *Jamaica Micro Xlet Project* as your project (see Figure 3.3).

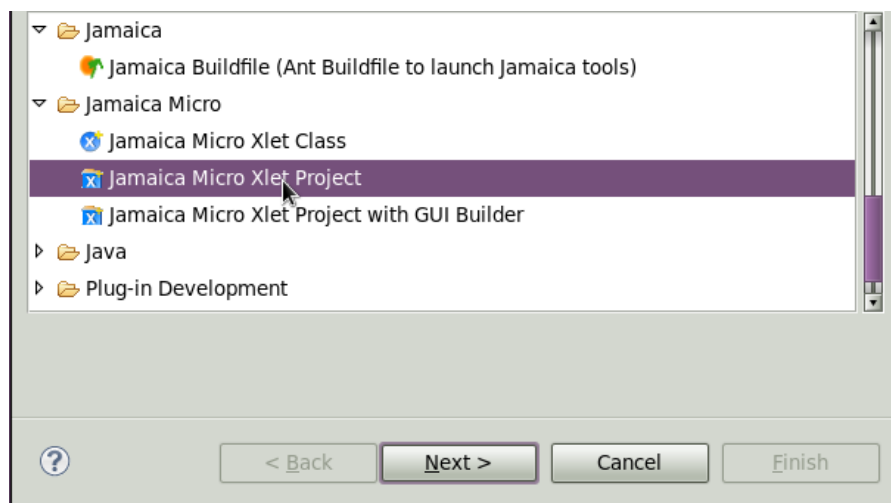


Figure 3.3: Jamaica Micro Xlet Project

3. Now name your project and select an Emulator (see Figure 3.4).

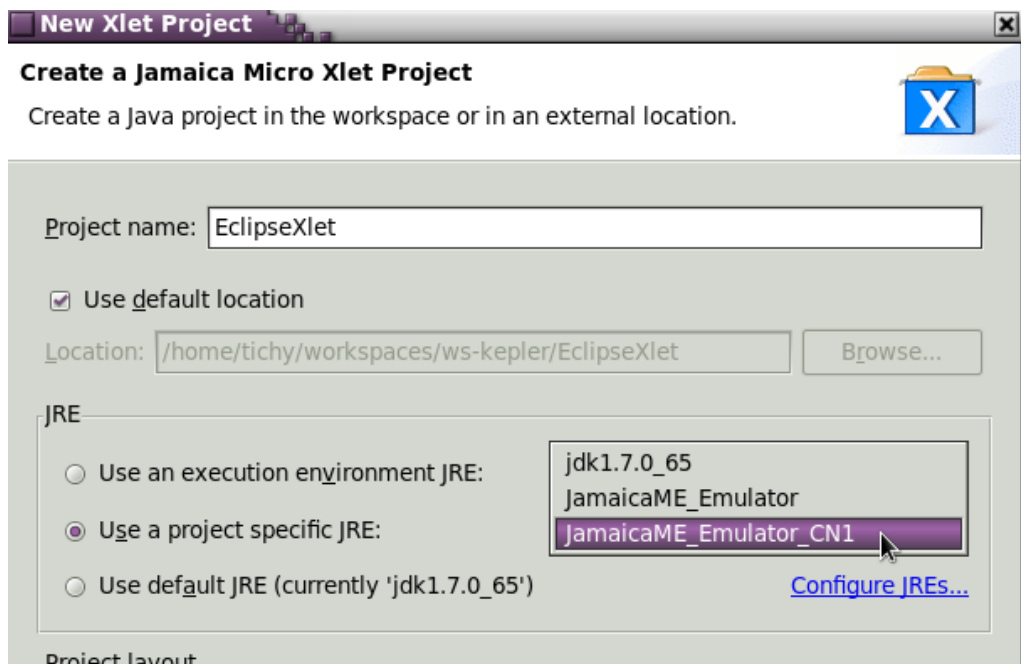


Figure 3.4: Selecting an Emulator

4. You have now created your Xlet project. However it does not yet contain a class, for this, create a >New>Other (as in step 1) Xlet Class in the src (source) folder (see Figure 3.5).

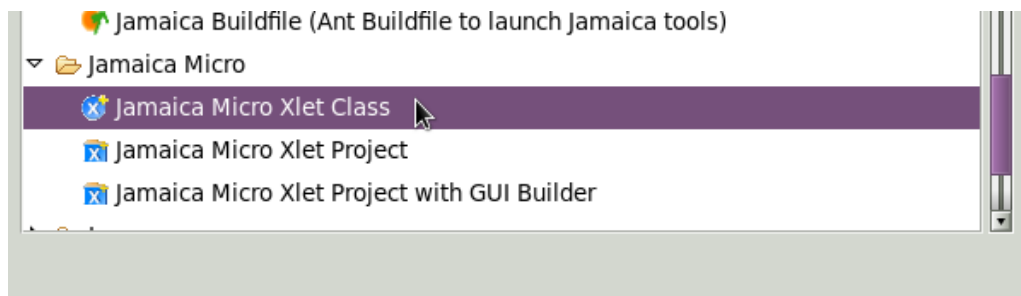


Figure 3.5: Jamaica Micro Xlet Class

5. Name your class and make sure it is in the right project and folder (see Figure 3.6).

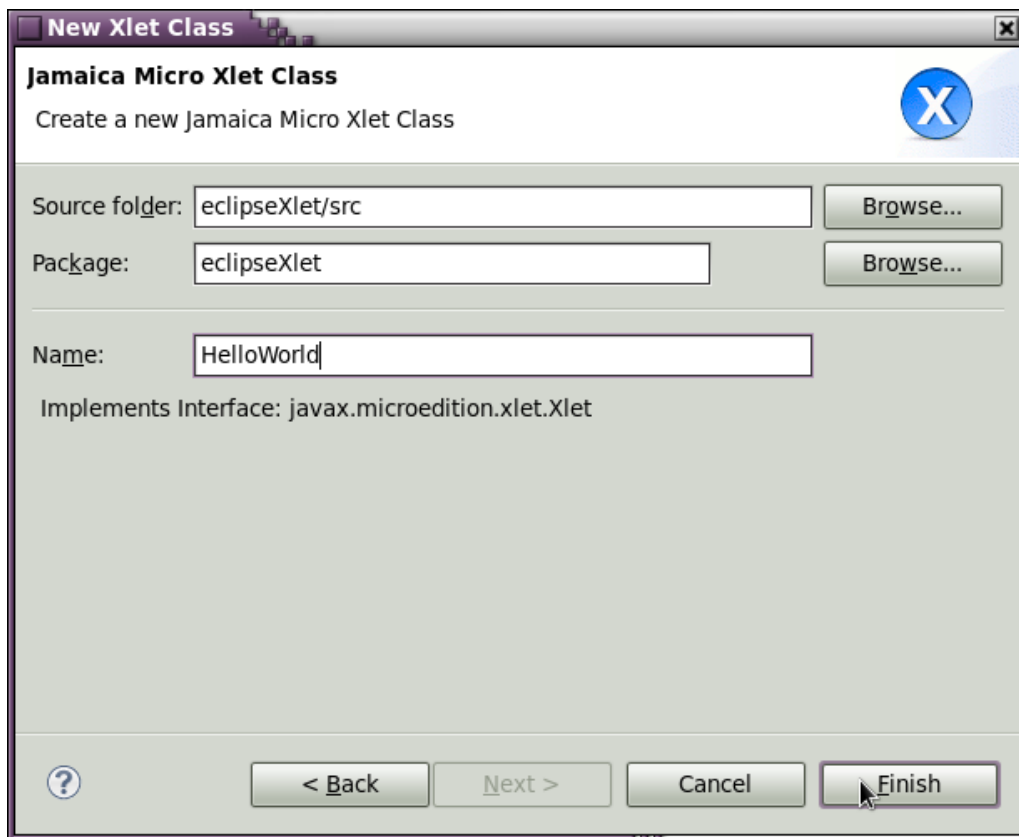
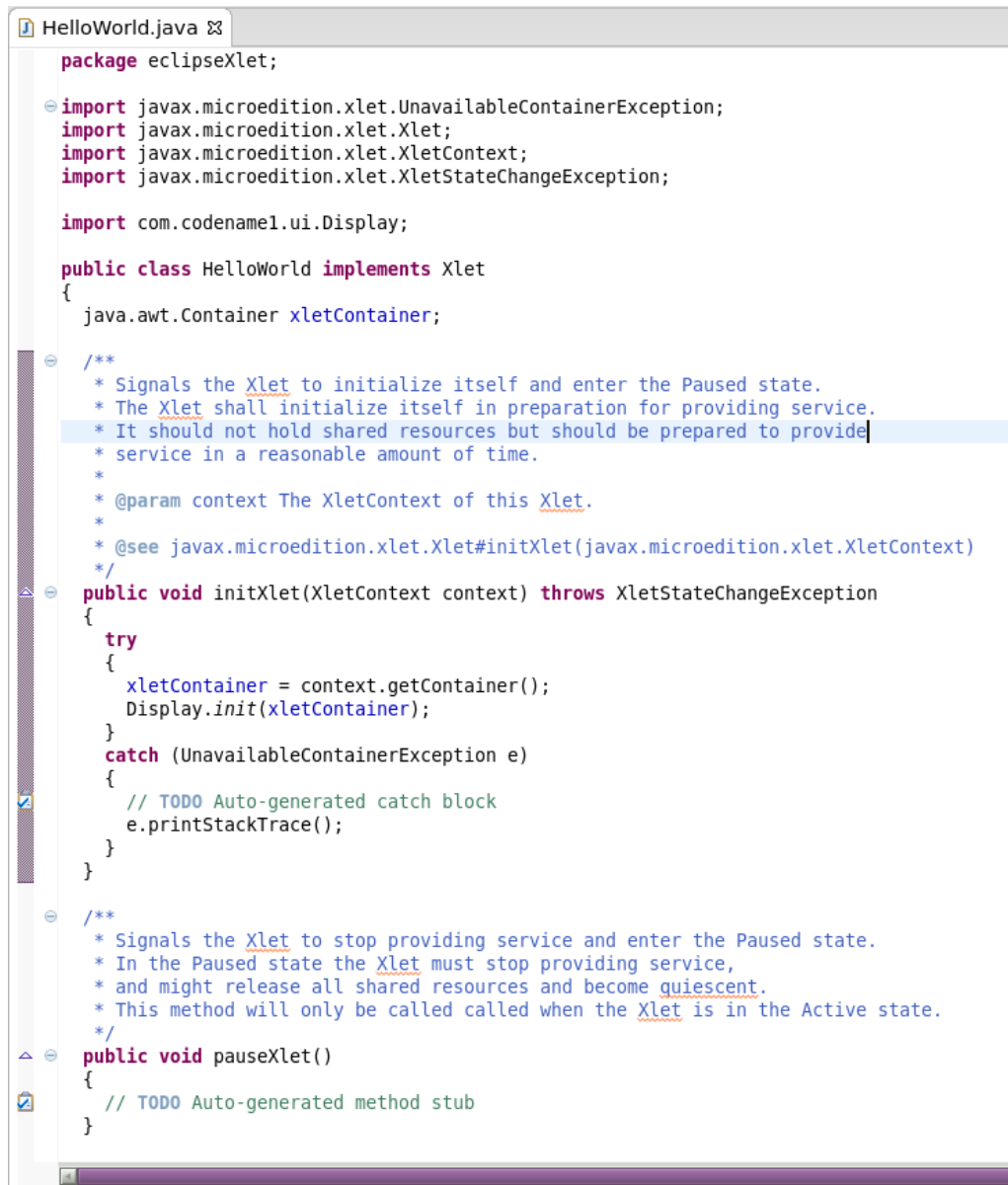


Figure 3.6: HelloWorld Class

6. You have now created your first Xlet class. If you open the class you will see that the basic framework of the Application is already given (Figure 3.7). See if you can spot the imported frameworks mentioned before.



```
package eclipseXlet;

import javax.microedition.xlet.UnavailableContainerException;
import javax.microedition.xlet.Xlet;
import javax.microedition.xlet.XletContext;
import javax.microedition.xlet.XletStateChangeException;

import com.codename1.ui.Display;

public class HelloWorld implements Xlet
{
    java.awt.Container xletContainer;

    /**
     * Signals the Xlet to initialize itself and enter the Paused state.
     * The Xlet shall initialize itself in preparation for providing service.
     * It should not hold shared resources but should be prepared to provide
     * service in a reasonable amount of time.
     *
     * @param context The XletContext of this Xlet.
     * @see javax.microedition.xlet.Xlet#initXlet(javax.microedition.xlet.XletContext)
     */
    public void initXlet(XletContext context) throws XletStateChangeException
    {
        try
        {
            xletContainer = context.getContainer();
            Display.init(xletContainer);
        }
        catch (UnavailableContainerException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

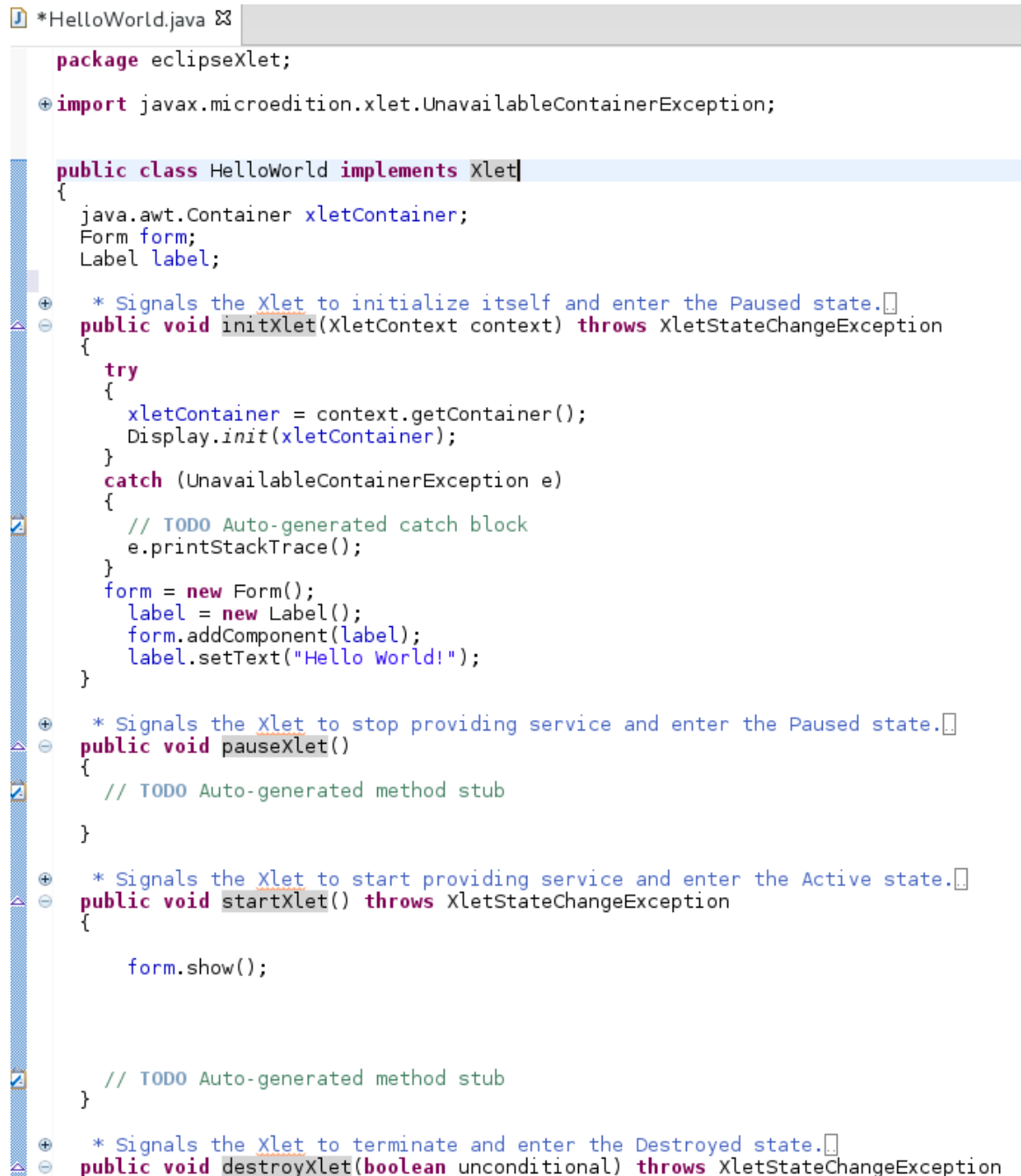
    /**
     * Signals the Xlet to stop providing service and enter the Paused state.
     * In the Paused state the Xlet must stop providing service,
     * and might release all shared resources and become quiescent.
     * This method will only be called when the Xlet is in the Active state.
     */
    public void pauseXlet()
    {
        // TODO Auto-generated method stub
    }
}
```

Figure 3.7: Xlet Class

3.3 Example: HelloWorld

Now you are all set to create your first Hello World application using Xlets. You can either import your elements from CodenameOne or from the outdated LWUIT 1.4 library.

Creating an application with Xlets may seem very confusing at first as the file already has code in it (see Figure 3.8). You are encouraged to first study the structure and compare it to what you know of Xlets. The basic structure is that there are four separate parts to the public class, these being the Xlet methods mentioned in Section 2.1.



```

*HelloWorld.java
package eclipseXlet;

import javax.microedition.xlet.UnavailableContainerException;

public class HelloWorld implements Xlet
{
    java.awt.Container xletContainer;
    Form form;
    Label label;

    * Signals the Xlet to initialize itself and enter the Paused state.
    public void initXlet(XletContext context) throws XletStateChangeException
    {
        try
        {
            xletContainer = context.getContainer();
            Display.init(xletContainer);
        }
        catch (UnavailableContainerException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        form = new Form();
        label = new Label();
        form.addComponent(label);
        label.setText("Hello World!");
    }

    * Signals the Xlet to stop providing service and enter the Paused state.
    public void pauseXlet()
    {
        // TODO Auto-generated method stub
    }

    * Signals the Xlet to start providing service and enter the Active state.
    public void startXlet() throws XletStateChangeException
    {
        form.show();

        // TODO Auto-generated method stub
    }

    * Signals the Xlet to terminate and enter the Destroyed state.
    public void destroyXlet(boolean unconditional) throws XletStateChangeException

```

Figure 3.8: First HelloWorld Xlet

Any integers, forms, labels, buttons etc., should be initialized in the public class so that all functions provided in the Xlet class can access them. In this case, both the Form and the Label are initialized at the beginning of the document, in the `initXlet` they are built, the text is set and everything is rendered visible in the `startXlet` method, hence the Form will only be visible in the Running state.

Having created a basic HelloWorld Xlet application, the question remains how to run the application from Eclipse using the emulator.

To run an application in the JamaicaCAR Emulator, it is necessary to create a new run configuration. Once you have configured it (compare Figure 3.9), you can easily run it again from the same menu as can be seen here, the name you gave the Configuration will be visible.

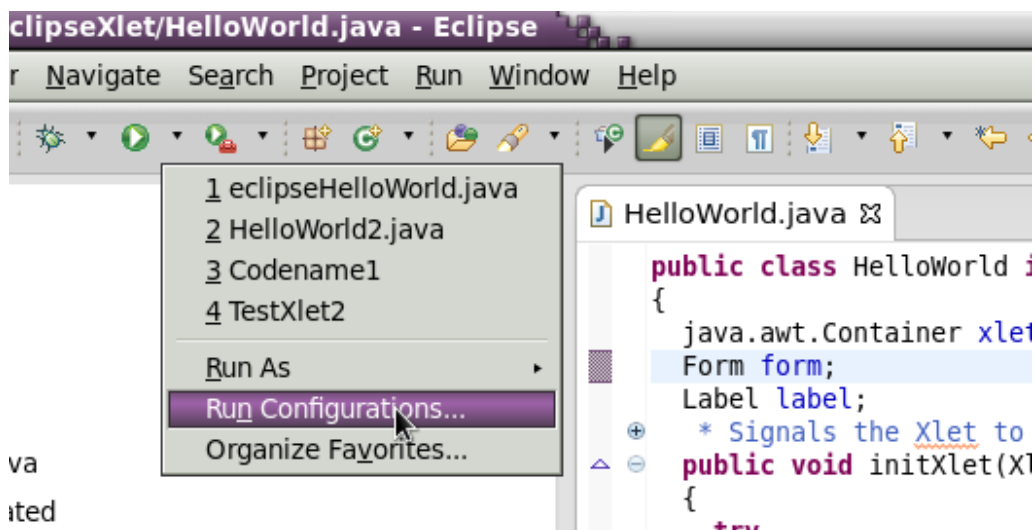


Figure 3.9: Making a new Run Configuration

Now you must select the desired project and the class within the project. You can either type the names manually or use the browse option or select it from the left-hand-side window. Running the application as a Java applet or Java application will not work, as they do not qualify as either. The Configuration must also be provided with a name (this will be the name that you will see in the run menu as in Figure 3.9).

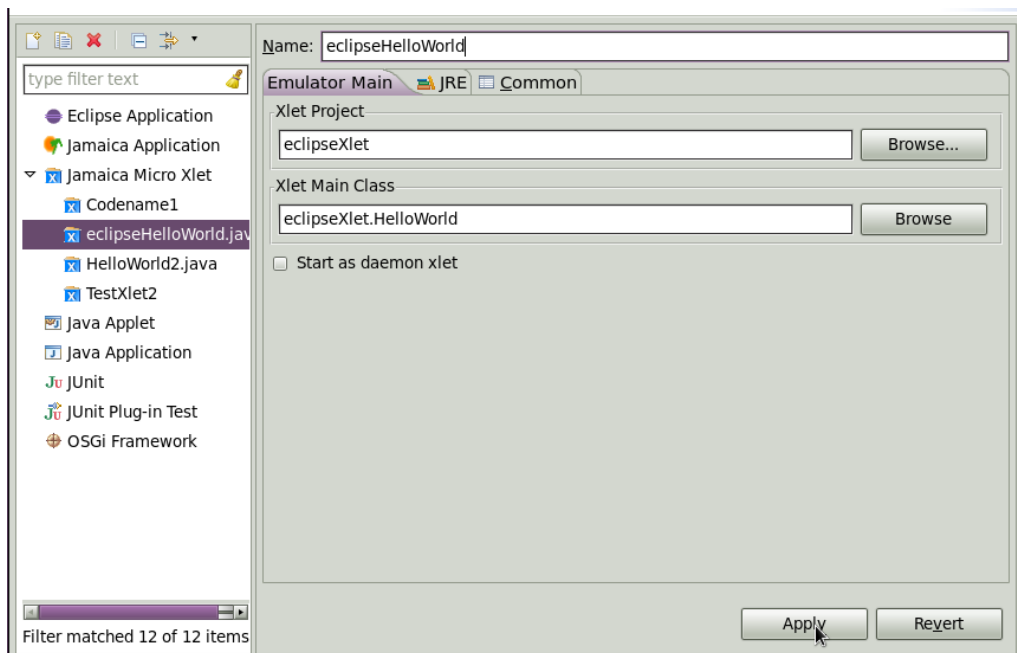


Figure 3.10: Applying a new Run Configuration

Once you have applied (Figure 3.10) your settings and run (Figure 3.11) the configuration, you will be able to view your application in the JamaicaCAR Emulator.

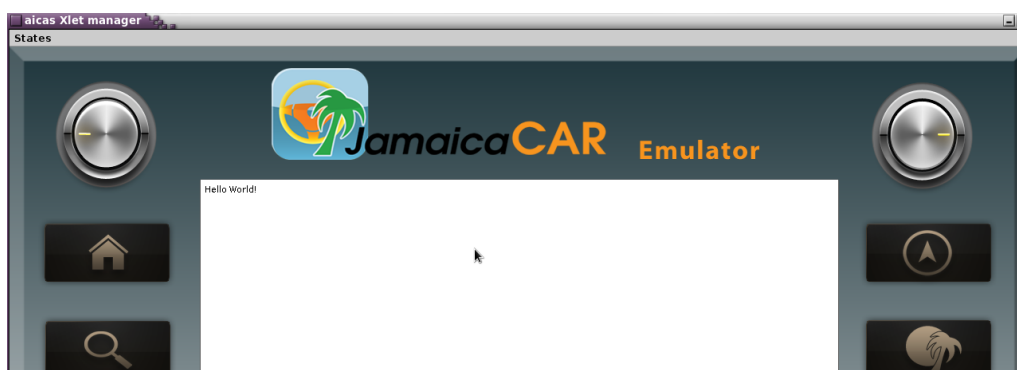


Figure 3.11: Running the Configuration in the Emulator

Chapter 4

Creating a Downloadable Application with the CodenameOne Designer

4.1 Initializing the Designer in Eclipse

The CodenameOne Designer is a tool from CodenameOne, with which a GUI is very easily created for an application. The following steps will show you how to get started with making a basic application using the CodenameOne Designer.

1. First, it is once again necessary to create a new project. Once again under File, choose >New>Other.
2. This time, choose the *Jamaica Micro Xlet Project with GUI Builder* as your Project (compare Figure 4.1).

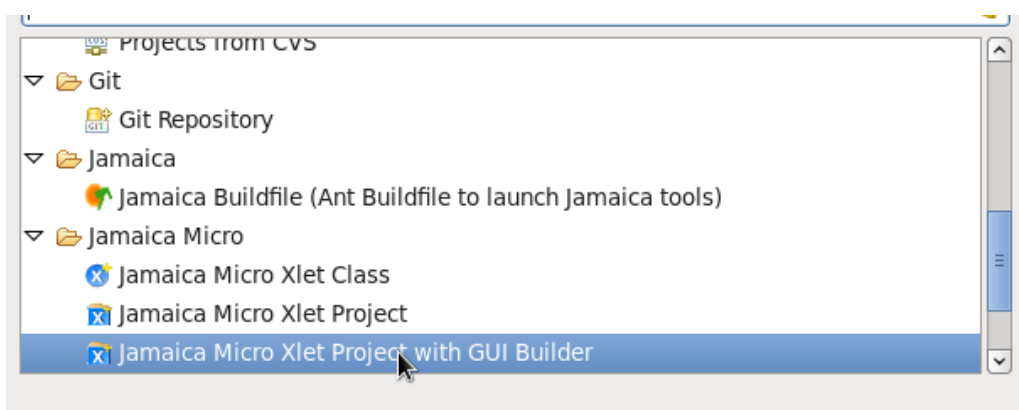


Figure 4.1: Use the 'Jamaica Micro Xlet Project with GUI Builder'

3. Name your project and change the JRE to a recent JamaicaCAR Emulator (Figure 4.2).

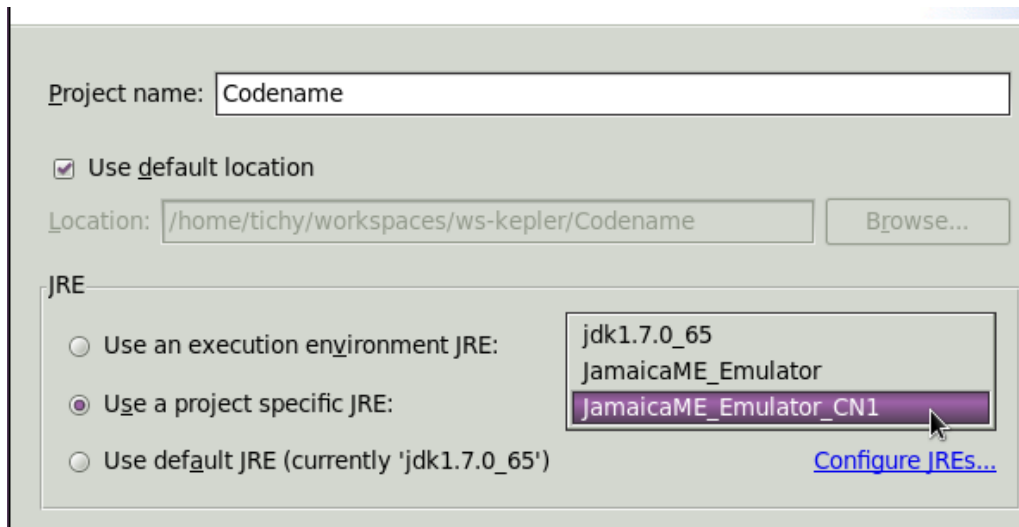


Figure 4.2: Use a recent JamaicaCAR Emulator

4. Name your Micro Xlet Class (Figure 4.3).

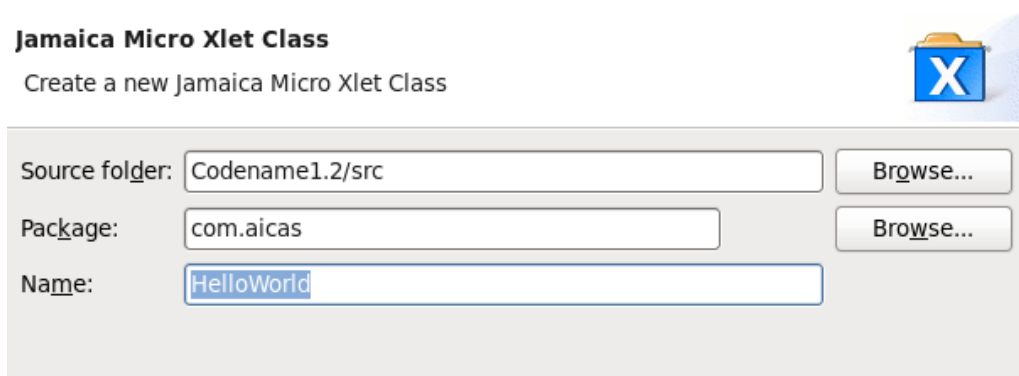


Figure 4.3: Create a HelloWorld Class

5. You have now created your CodenameOne Designer Project. In order to open the Designer, double click the GUI.res file (Figure 4.4).

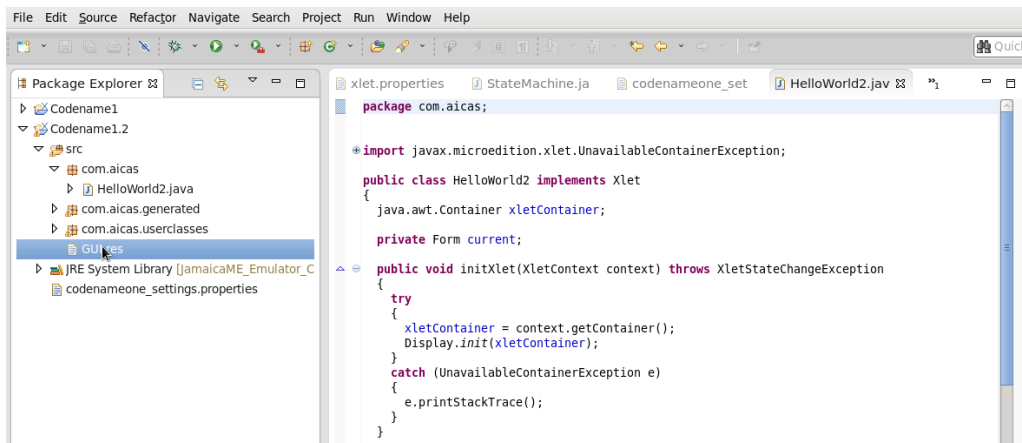


Figure 4.4: GUI.res

6. Finally, select the 'main' GUI element to see the Designer core components and your current basic HelloWorld application which is automatically generated (Figure 4.5).

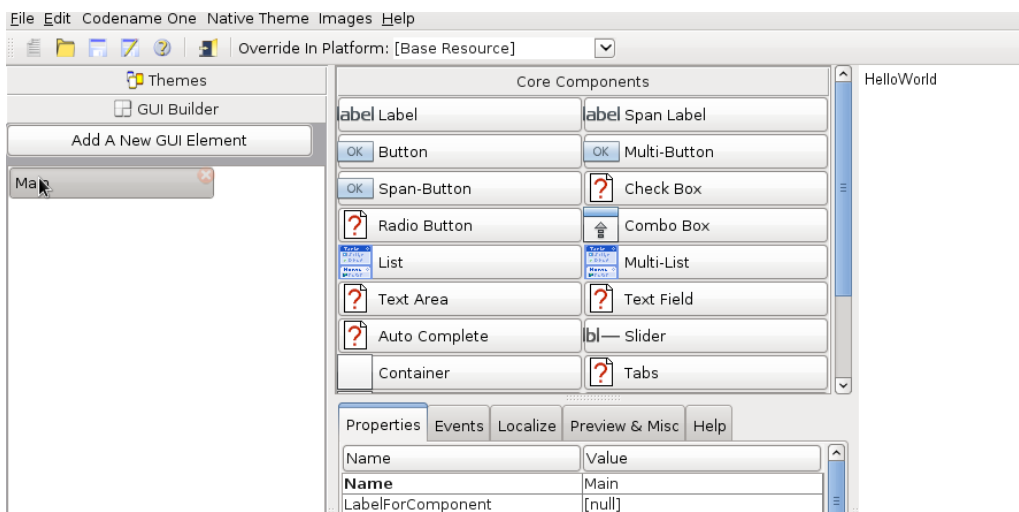


Figure 4.5: CodenameOne Designer Main

4.2 Designer basics

The many components in the Designer make programming a graphical application much easier. Using the Designer, it becomes incredibly simple to add buttons and labels and any number of components to the application; in order to add components to the project, simply drag them into the interface.

4.2.1 Action Listeners

If a 'Button' is added to the application (point and drag a Button Component from Components list to the interface), it would be best if it also had an effect, for this an action listener must be added to the component. In order to create an action listener on a component, it is necessary to select 'Events' and, with the desired component selected, press 'Action Event' (see Figure 4.6). The selected component will become bold in the overview below. It is however not possible to determine the reaction to an action event in the CodenameOne Designer itself, for this it is necessary to write code in the application by using Eclipse. More on this in the Section 4.2.2.

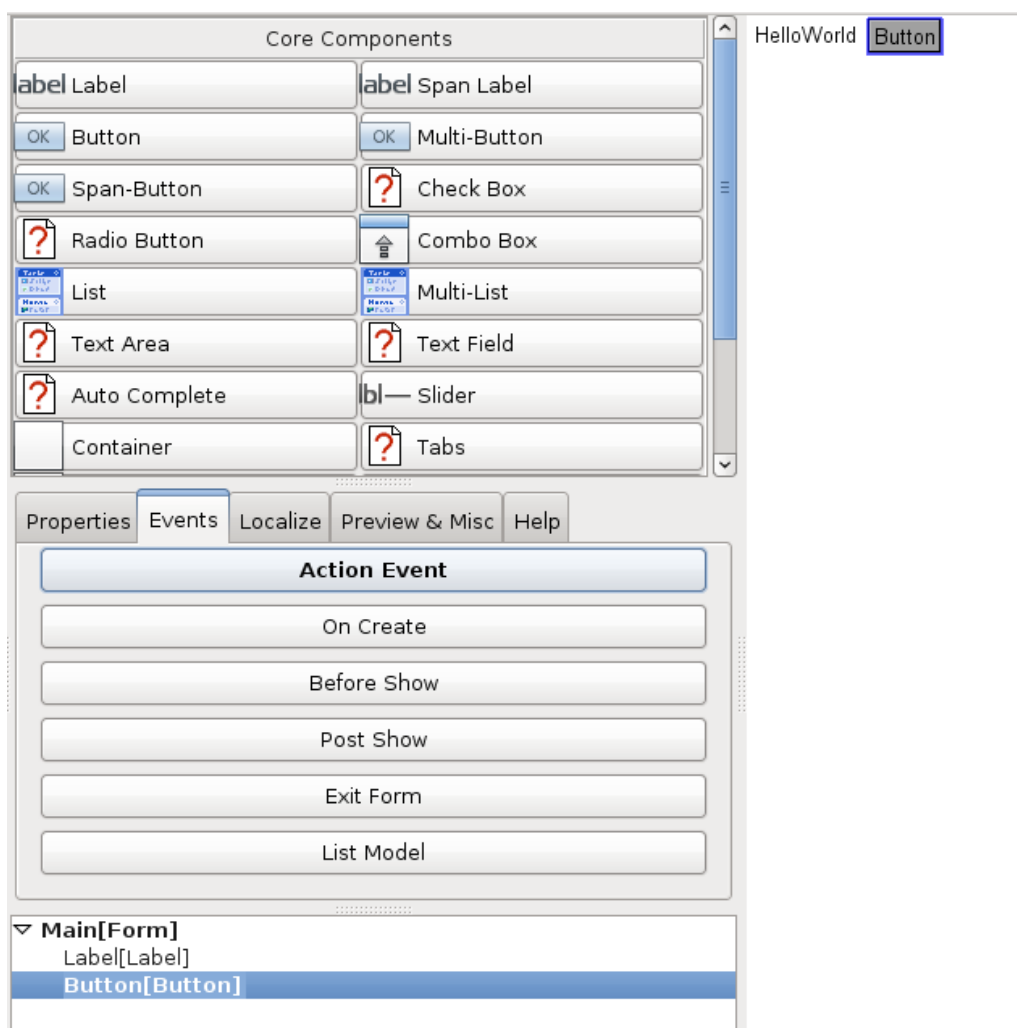


Figure 4.6: Action Listener with the CodenameOne Designer

Running the application generated using the CodenameOne Designer works in the same way as with Eclipse. See Section 3.3.

4.2.2 Action Event

The reaction to an action listener must be written in the `StateMachine.java` class. This class can be found in the source folder in the `com.aicas.userclasses` package. In this example (compare Figure 4.7), a GUI has been built with a button that has an action listener.

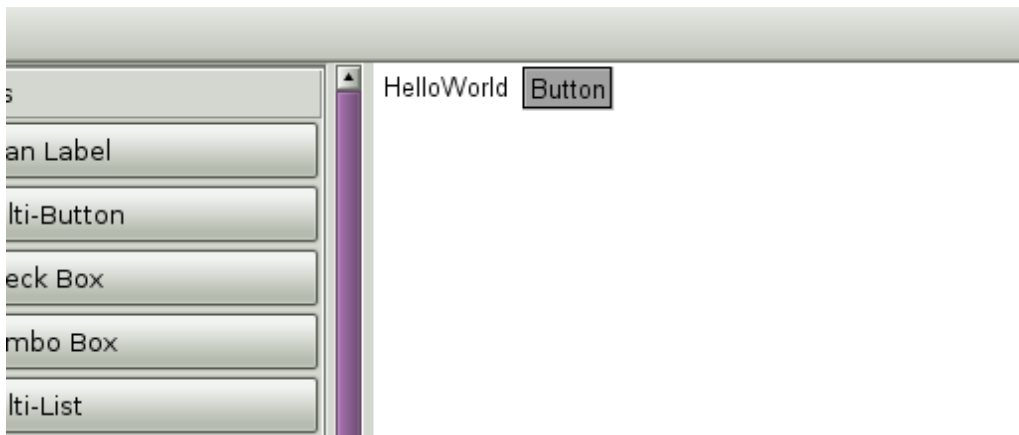


Figure 4.7: GUI with a Button with an Action Listener

The `StateMachine` files are automatically generated by the Designer. They are the code version of that which you built in the Designer (the buttons and labels and containers etc.).

In the `StateMachine.java` file, the desired event can be written in the `ButtonAction` method. This method is found at the very bottom of the `StateMachine.java` file, you needn't worry about any of the other code in the file. In this case, the event will be to change the background color of the entire Display. This is done by, on an instance of the action listener (the button was pressed), checking the current style's `BgColor` and changing it to the color `0xfe5498`. This is followed by repainting the display in its style which then has a different setting (see the code in Figure 4.8).

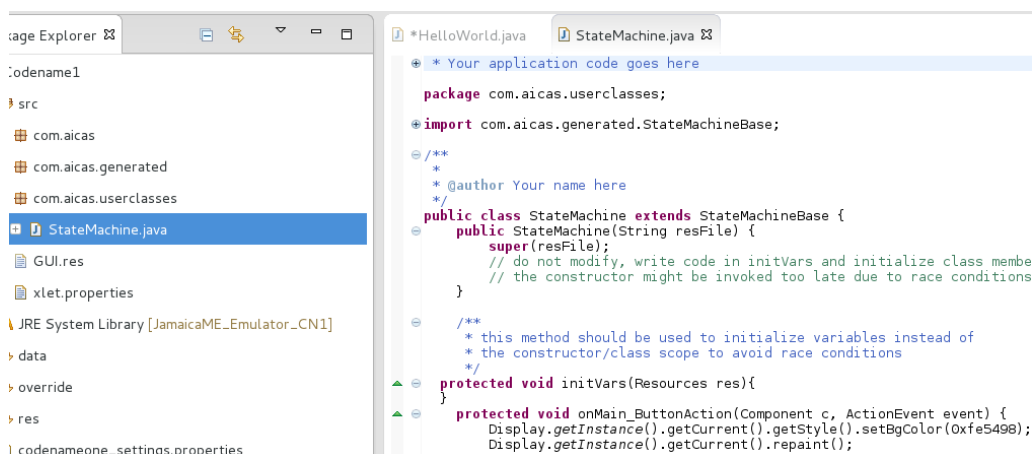


Figure 4.8: Action Event written in `StateMachine.java`

Now, when the application is started in the Emulator and the button is pressed...
...the color changes to pink!

4.2.3 Containers

It is of course possible to make more complicated UIs. A key component that is essential for applications that consist of more than one button is the Container. This has the ability to, as the name suggests, contain other Components and, recursively, other Containers as well. There are multiple types of Components, what Components exist and which Containers can contain which Components is displayed very effectively in Figure 4.9.

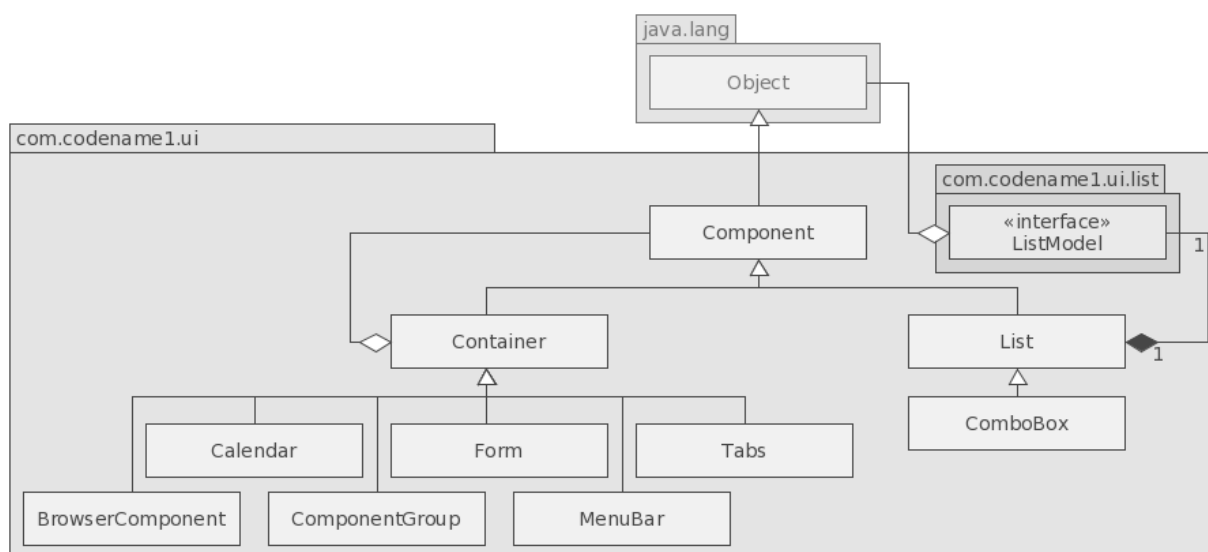


Figure 4.9: Container Hierarchy

4.2.4 Layouts

Containers also enable an effective and easy way to arrange the components enclosed in them. In the properties window in the Designer, when a container or the main form is selected, there is an option called 'Layout' (see Figure 4.10, where we have a Button, a Container that is containing nine RadioButtons, and a Text Area).

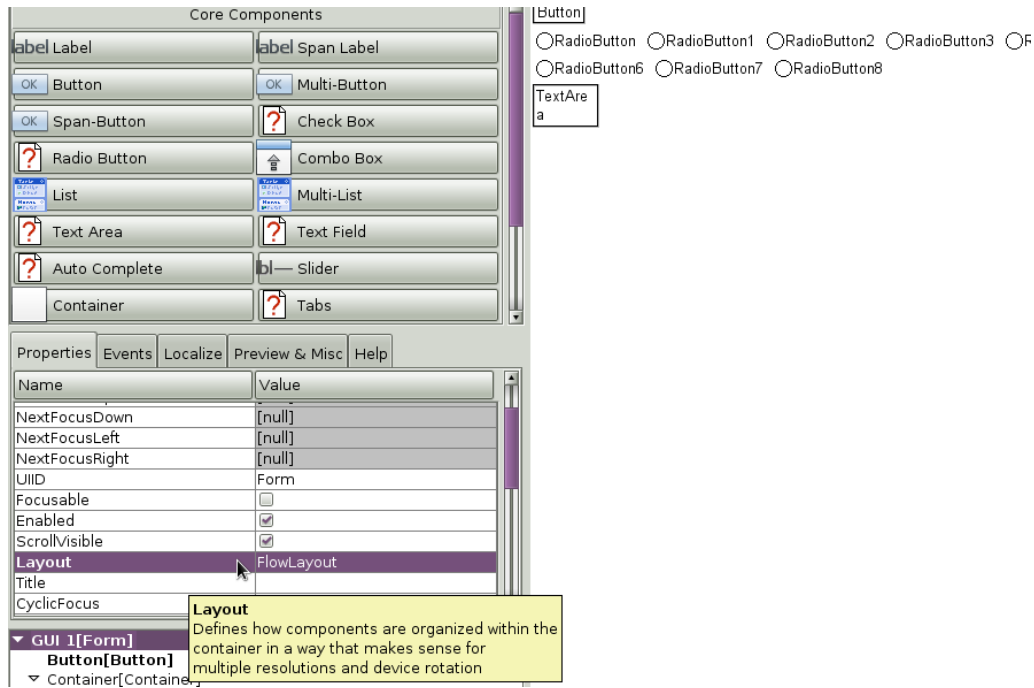


Figure 4.10: Form Layout

When selected, a Layout manager opens where you can choose between multiple different layouts. A model of this is shown in the right-hand side window in Figure 4.11. In this case, the 'Border Layout' will be used.

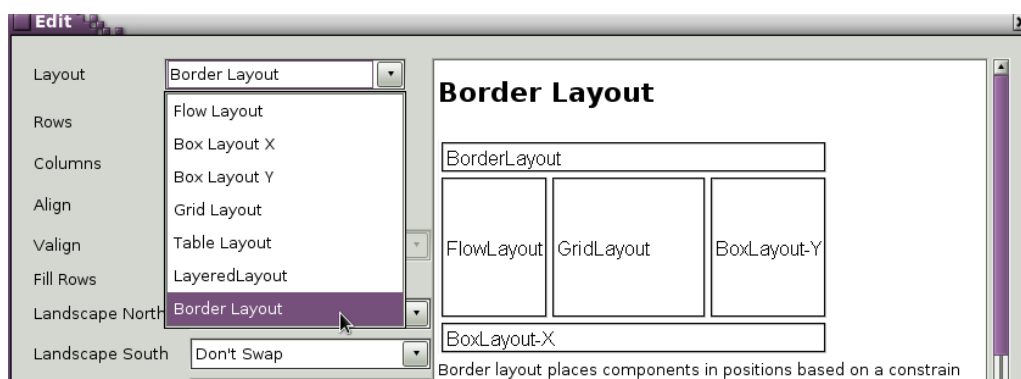


Figure 4.11: Border Layout

As you can see in the background of Figure 4.12, the Button, the TextArea and the Container have already been rearranged. Now the layout within the Container will be modified to be a 'Grid Layout' with 3 rows and 3 columns.

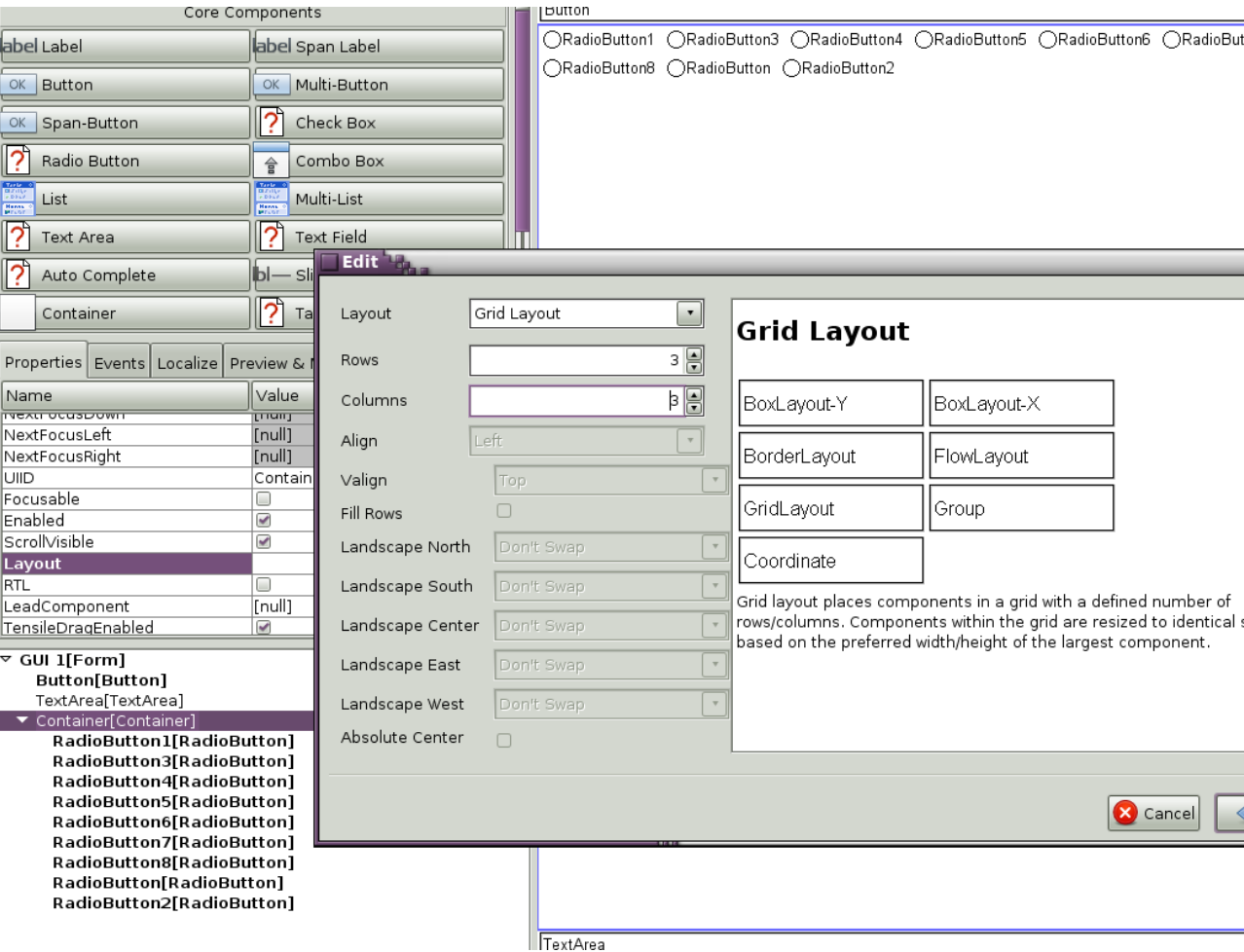


Figure 4.12: Layout in the Container

There are of course many more layouts available. There are: Flow, Box X or Box Y, Grid, Table, Layered and Border Layouts. Try them out and experiment which would suit your application best.

As a result of the manipulation, the RadioButtons have automatically been arranged in a grid and we now have a GUI where the general Layout is a Border Layout but another, different, Layout has been applied to a Container in the main Form (see Figure 4.13).

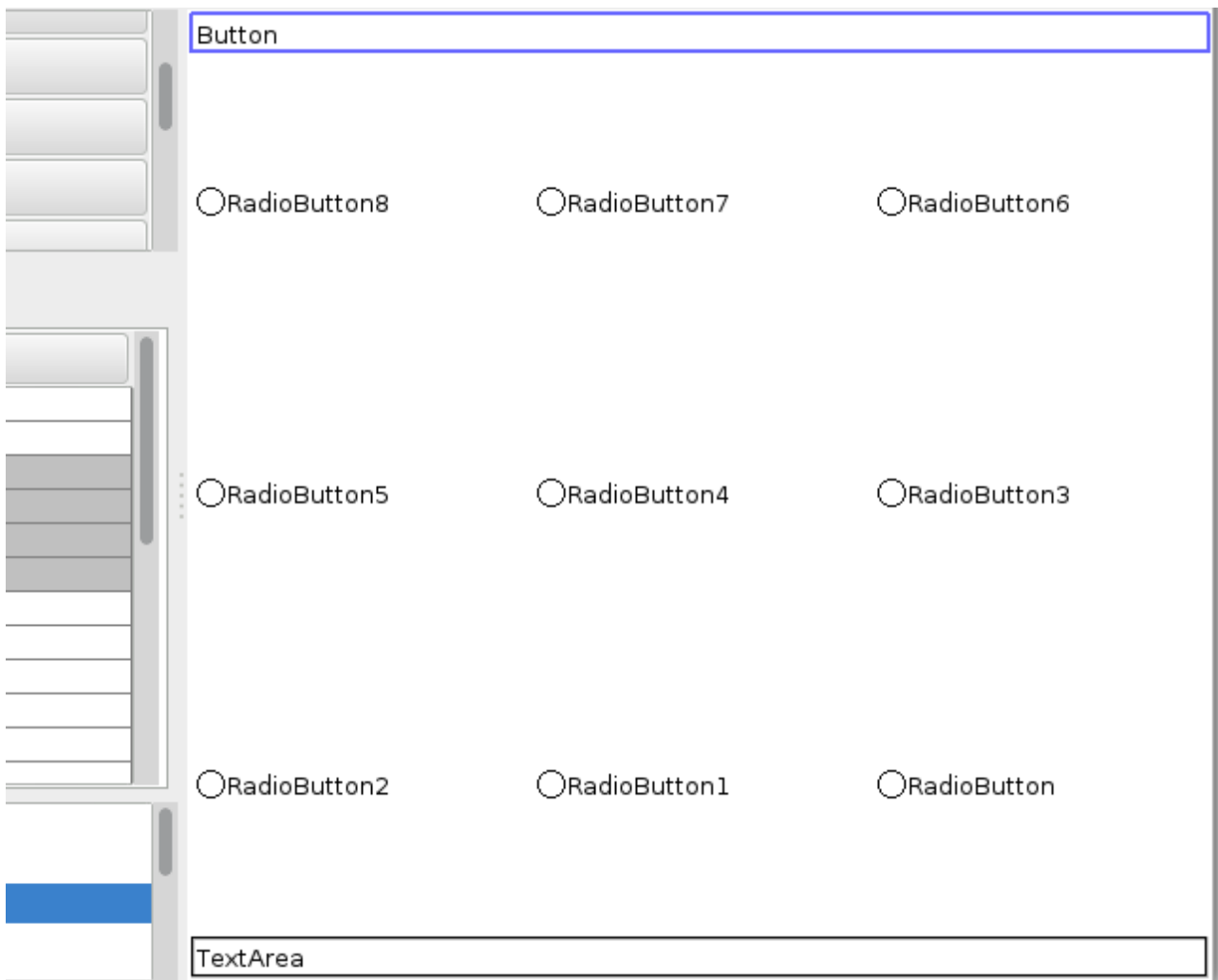


Figure 4.13: Grid Layout

4.2.5 Themes

For those people who do not want their application to consist of a white background with grey buttons; CodenameOne provides 'Themes'. While it is possible to change the color of things (as we saw before) and also to change the shape, why should the user have to spend so much time designing components when the CodenameOne Designer does it for you?

The Themes provided by the Designer change the look of your application by providing a skin for your application, sort of like a background image. These are situated just above the GUI Builder tab on the left-hand-side of the Designer. Selecting 'Add A New Theme' will allow you to choose from a variety of eloquent designs for your application (see Figures 4.14 and 4.15).

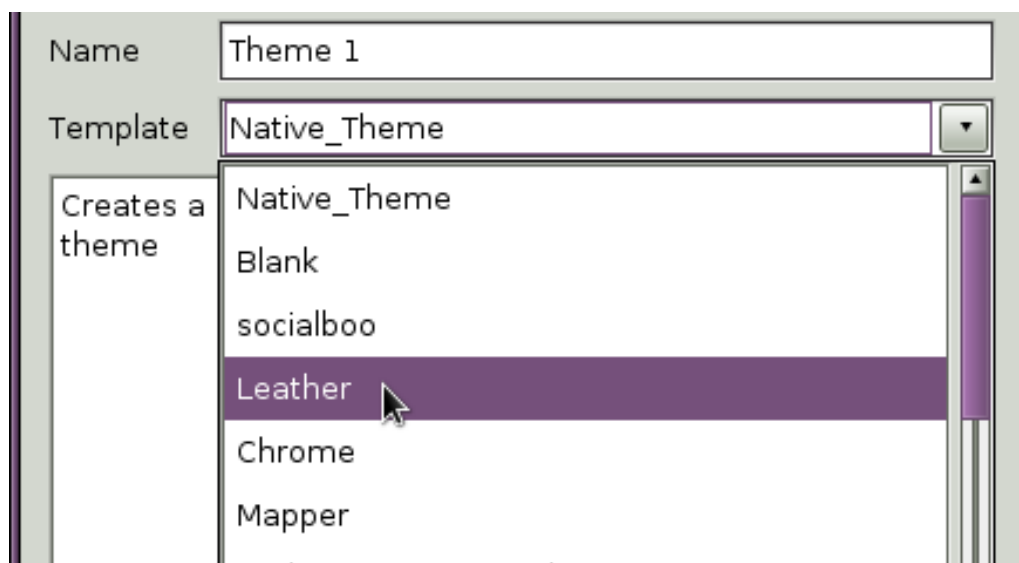


Figure 4.14: Selecting a Theme

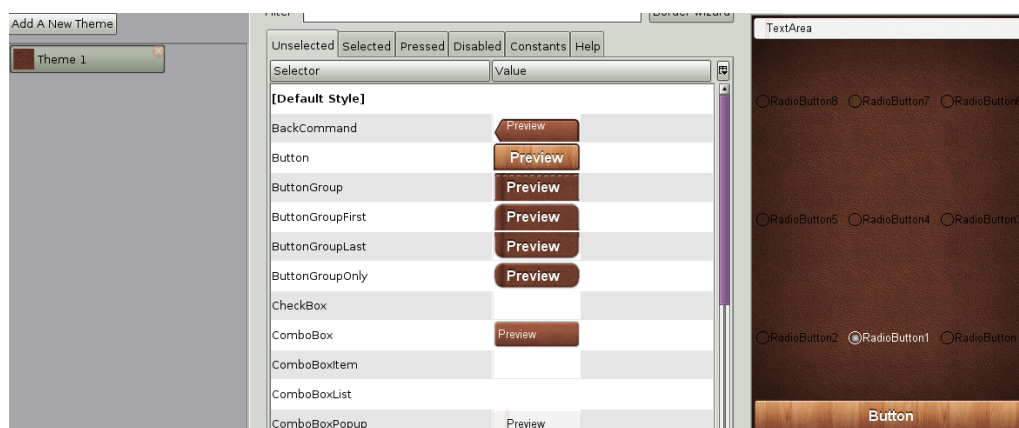


Figure 4.15: GUI with a Theme

If you are having trouble getting a theme to affect a certain GUI, check if, in the Theme tab of the Designer, the desired GUI is selected just above the preview.

Changing the style's `BgColor` as in the the HelloWorld example will not change the color of the Theme. The `BgColor` will change, but because you have added a Theme, the background is no longer visible, it is, so to say, behind the theme. Hence you cannot see a color change.

There are countless more options to explore in the CodenameOne Designer, you are now ready to design your own applications, according to your level of Java experience without major technical difficulties.

Appendix A

Further Reading

- <https://www.aicas.com/cms/en/JamaicaCAR>
- http://www.interactivetvweb.org/images/tutorials/common/xlet_state_machine.gif
- <http://docs.oracle.com/javase/tutorial/deployment/applet/>
- <http://www.codenameone.com/developer-guide.html>
- https://www.aicas.com/cms/sites/default/files/jamaicavm_6.3_manual.pdf
- <https://wiki.aicas.burg/wiki/images/d/d5/AMSManual.pdf>