CREATE AN UBER CLONE IN 7 DAYS

BUILD A REAL WORLD FULL STACK MOBILE APP IN JAVA

SHAI ALMOG

Create an Uber Clone in 7 Days

BUILD A REAL WORLD FULL STACK MOBILE APP IN JAVA

Shai Almog
Codename One Academy

Contents

8 Day 4: Search Route and Hailing

9 Day 4: Search Route and Hailing – Night Hack

10 Day 5: Driver App and Push

11 Day 6: Billing and Social Activation

Preface *

Last year we launched the Codename One Academy. As part of that offering we surveyed the Codename One community and asked them: *“what would you like to learn?”*.

The response was was overwhelmingly: *“How to build an app like Uber!”*.

At first I thought about creating something in the style of Uber but I eventually settled on building something that looks really close to the native app. Almost a clone.

My motivation for going for a clone instead of coming up with a completely new design was driven by this line of thinking:

- I wanted the design to look professional and you can’t go wrong with a design from a top tier vendor
- People can learn a lot by understanding the decisions Uber made—I know I did
- If I would have built something different I might have given myself “discounts” that don’t exist in the real world

I used the word clone to indicate the similarity but not to indicate a carbon copy. Uber is a huge and nuanced app and I had only one week to write all the relevant applicable code.

My goal was to do the “hard stuff” and gloss over some of the deeper details. The goal is to teach with a strong focus on the mobile side. I wanted to create a book that would show you how to build a fully functional MVP (Minimum Viable Product) within a week. I wanted to illustrate the shortcuts that make sense and those that don’t. This is a powerful approach whether you are building a startup or working within a large corporation.

I think developers can’t deliver truly innovative ideas if we are constantly doing the same app over and over. By making this process simpler I hope that developers will adopt innovative ideas faster rather than re-do the same apps all over again.

Audience

My bookshelf is overflowing with programming books. Most of them revolve around teaching a specific technology (e.g. Java). A few discuss architecture or other big concepts. None of them teach how to build the whole thing.

These books demonstrate through small localized samples. The results don’t look like a professional production app. They skip details like servers and business logic or client UI nuances. I didn’t want to write that book. This book tries to address the whole thing, the full stack. Even if you don’t want to become a full-stack developer, understanding the whole picture is often helpful.

Prerequisites

If you think you have basic understanding of the following you should be able to follow this book:

- Java – a basic/intermediate Java level should be enough. You will need familiarity with one of the top 3 Java IDE’s (NetBeans, IntelliJ/IDEA or Eclipse)
- Maven – basic understanding we won’t do anything too fancy
- REST/JSON – we will create JSON based web services with HTTP GET/POST methods. This is explained in the book but I don’t explain HTTP GET/POST or JSON

> This book is “code heavy” due to its nature, if you have a difficulty reading listings this book might be difficult to follow

What you don’t need to Know

While it goes without saying that we don’t need to know everything. I wanted to clarify some specific details you don’t need to know:

- Codename One - the book teaches the parts of Codename One that are applicable to the book
- Spring Boot - we’ll discuss the parts of Spring Boot that matter. This isn’t a book about Spring Boot so I won’t get too deep into that but you should be able to work with it after reading the book
- SQL/MySQL/JPA - MySQL is used for storage in the book and we abstract it via JPA. While you will need to install MySQL you won’t need to know SQL or how to work with it as we’ll use JPA. I will cover JPA at a high level within the book while discussing the elements of interest

Get this Book for FREE!

This book contains the Uber clone part of the online course “Build Real World Full Stack Mobile Apps in Java” available here: codenameone.teachable.com/p/build-real-world-full-stack-mobile-apps-in-java.

That course is **much** bigger and constantly growing. Currently it covers the process of creating a Facebook Clone, a restaurant menu application and will cover more apps in the near future (e.g. a WhatsApp Clone). The Uber clone portion of the course is over 5 hours of videos and presentations.

The total amount of materials in the course contains more than 15 hours of materials that grow on a monthly basis!

The course will also include the full book as part of the Uber clone module once exclusivity expires. If you bought this book you can use the coupon code `[REDACTED]` to get a 10% discount on the price of the course.

> This offer is limited and will expire on October 31st 2018!

How to Read this Book?

Most tech books allow you to skip ahead or just browse through the index to find what you are looking for.

That might not work as effectively with this book. This book describes a **real world app** and as such you might find it difficult to skip ahead. By its nature **the book is "code heavy"**.
That's unavoidable due to the basic premise of the book.

Some materials that are more general purpose were placed in appendices to make them more accessible.
The first few chapters prepare you for the journey ahead. The rest of the book is divided into seven days and three night-hacks. This division matches my experience in prototyping and building similar projects over the years.

You can skip the first chapters if you are familiar with the materials within, however I suggest paying special attention to the styling portion. In the styling section I explain the style syntax I used through the entire book.

I divided the days so each day fills in a different piece of the puzzle:

- After day one you can run a mockup of the client side UI. You can even run it on the device and it will all work. I find that having something you can "touch" is a huge motivator so I always start with the UI
- After day two the backend server will work and run
- After day three the server and the client will work with each other and you will be able to connect from the client to the server
- After day four you will be able to search for a location and see a route
- After day five hailing will work and you'll have the second app for the driver side
- After day six billing will be plugged in and you'll be able to login with Facebook or Google
- After day seven settings will work, transitions and animations would be more refined

Software Prerequisites

You will need JDK 8 (Java 8) to run the current code in the book. Notice that at this time Codename One doesn't support JDK 9 but this will probably change before 2019.

You will need a Java IDE: NetBeans, IntelliJ/IDEA or Eclipse. With the Codename One plugin installed from the Codename One website. Check out Appendix A for more details on setting up Codename One.

Getting Help

This book is dense with information and listings. It also mixes concepts from several tiers into one relatively short book. It's easy to miss a detail and it's probable I missed details when writing this book.

I'm always here to answer your questions, just ask a question in the Codename One discussion forum: www.codenameone.com/discussion-forum.html

Or on StackOverflow with the `codenameone` tag: stackoverflow.com/tags/codenameone

> I prefer StackOverflow but their moderators can be unwelcoming for some question types

Using the Code

The code for the hello world and TODO app built in the first two chapters is available for download here: www.codenameone.com/files/HelloWorldAndTodo.zip

The code for the Uber clone application build throughout the book is available to download here: [REDACTED...]

You may use the code from this book freely for any purpose with no restrictions or attribution. You can provide attribution if you wish to do so (and it would be appreciated).

Since it's code and there is no way to verify it I can't require that you purchase the book (or the online course) to use the code. Doing so would require restrictions that would potentially impact people who bought the book and I don't want to do that. So please consider this a moral imperative, if you make use of the source please buy the book or the course.

Important Notice About Cloning and Copyrights

Uber ™ is a trademark of Uber Technologies Inc.

This work is intended strictly for educational purposes. We don't condone the misuse of Uber's intellectual property!

The goal of this book is to teach via familiarity. Since the Uber application is well designed and familiar we chose it as our target but the book isn't meant as a "copy Uber" cookbook.

Many applications are built around ideas similar to Uber and utilize designs inspired by Uber. It's our assumption that you can learn a lot by understanding how to build something "like" Uber.

In this case we make use of Uber copyrighted work under "fair use" for teaching purposes. Shipping an application with the exact designs/logos or any similar markings goes against copyright law and might get you in trouble. That is why the demos in this book aren't available on the appstores. They would be illegal to ship.

This book is intended as a homage to Uber and their bold UI choices. As I wrote this book I developed a deep sense of respect to the nuanced work of the team that built the Uber app and I hope this is conveyed within the book.

Thanks and Acknowledgments

This work wouldn't have been possible without the immense help I got from **Chen Fishbein** and **Steve Hannah**. Both of whom supported the process of the books development throughout. Steve practically edited the book and deserves co-author credit!
This book literally wouldn't exist without both of them!

I'd also like to thank the reviewers whose feedback improved this book immensely, I would thank you each personally by name but since early reviews were anonymized I don't have access to your names!

Thank you for taking the time to read the earlier rough drafts and provide valuable feedback that undoubtedly improved this book immensely.

I do have a few names for late stage reviewers who sent feedback and words of encouragement. Special thanks goes out to: Francesco Galgani, Rémi Tournier & Steve Nganga.

Hello World 1

This chapter covers:

- What is Codename One?
- Creating a hello world application
- Signing a mobile application and building the native app
- Core concepts of mobile development, why mobile is different

I didn't teach my kids swimming by throwing them in the pool (that's my story and I'm sticking to it). But I think it's a wonderful way to teach a new technology so we'll start with a trivial hello world to understand the basics. I tried to write a fluent book that doesn't burden the reader with every detail but this is a tight rope to walk. I've listed further resources in the end of the chapter if you need further information.

What's Codename One? 1.1

Codename One is a Write Once Run Anywhere mobile development platform for Java/Kotlin developers. It integrates with IntelliJ/IDEA, Eclipse or NetBeans to provide seamless native mobile development.

The things that make it stand out from other tools in this field are:

- Write Once Run Anywhere support with no special hardware requirements and 100% code reuse
- Compiles Java/Kotlin into native code for iOS, UWP (Universal Windows Platform), Android and even JavaScript/PWA
- Open Source and Free with commercial backing/support
- Easy to use with 100% portable Drag and Drop GUI builder
- Full access to underlying native OS capabilities using the native OS programming language (e.g. Objective-C) without compromising portability
- Provides full control over every pixel on the screen
- Lets you use native widgets (views) and mix them with Codename One components within the same hierarchy (heavyweight/lightweight mixing)
- Supports seamless Continuous Integration out of the box

Codename One can trace its roots to the open source LWUIT project started at Sun Microsystem in 2007 by Chen Fishbein (co-founder of Codename One). It's a huge project that's been under constant development for over a decade!

As such I'll only scratch the surface of the possibilities within this book.

1.1.1 *Build Cloud*

One of the things that make Codename One stand out is the build cloud approach to mobile development. iOS native development requires a Mac with xcode. Windows native development requires a Windows machine. To make matters worse, Apple, Google and Microsoft make changes to their tools on a regular basis…

This makes it hard to keep up.

When we develop an app in Codename One we use the builtin simulator when running and debugging. When we want to build a native app we can use the build cloud where Macs create the native iOS apps and Windows machines create the native Windows apps. This works seamlessly and makes Codename One apps native as they are literally compiled by the native platform. E.g. for iOS builds the build cloud uses Macs running xcode (the native Apple tool) to build the app.

> Codename One doesn't send source code to the build cloud, only compiled bytecode!

Notice that Codename One also provides an option to build offline which means corporations that have policies forbidding such cloud architectures can still use Codename One with some additional overhead/complexity of setting up the native build tools. Since Codename One is open source some developers use the source code to compile applications offline but that's outside the scope of this book.

For a more thorough explanation of the underlying architecture and principals of Codename One check out Appendix F (page 415).

1.2 *Getting Started*

The following instructions assume you installed the Codename One plugin into your IDE. If you didn't do that you can check out the install instructions in Appendix A (page 389).

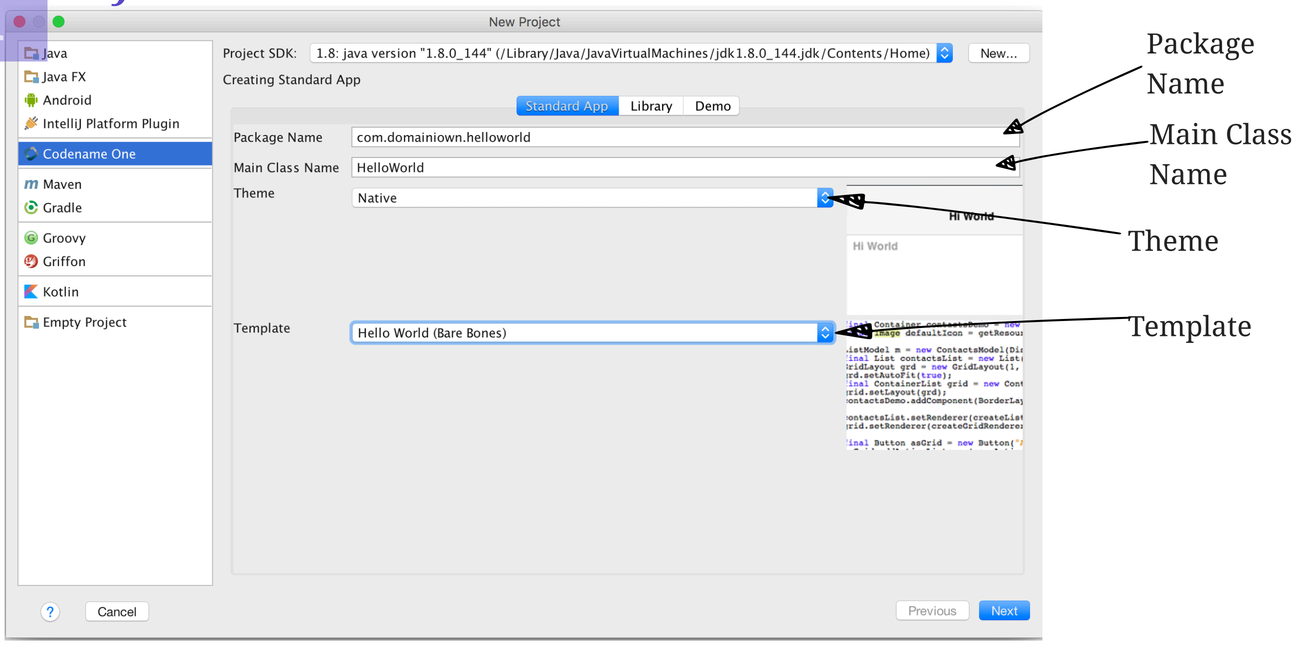
New Project

Before we get to the code there are few important things we need to go over with the new project wizard.

We need to create a new project. We need to pick a project name and I'll leave that up to you although it's hard to go wrong with `HelloWorld`. The following four values are important:

- **App Name** - This is the name of the app and the main class, it's important to get this right as it's hard to change this value later
- **Package Name** - It's **crucial** you get this value right. Besides the difficulty of changing this after the fact, once an app is submitted to iTunes/Google Play with a specific package name this can't be changed! See the sidebar Picking a Package Name
- **Theme** - There are various types of builtin themes in Codename One, for simplicity I pick `Native` as it's a clean slate starting point
- **Template** - There are several builtin app templates that demonstrate various features, for simplicity I always pick `Bare Bones` which includes the bare minimum

IntelliJ


New Project

Project SDK: 1.8: java version "1.8.0_144" (/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home)

Creating Standard App

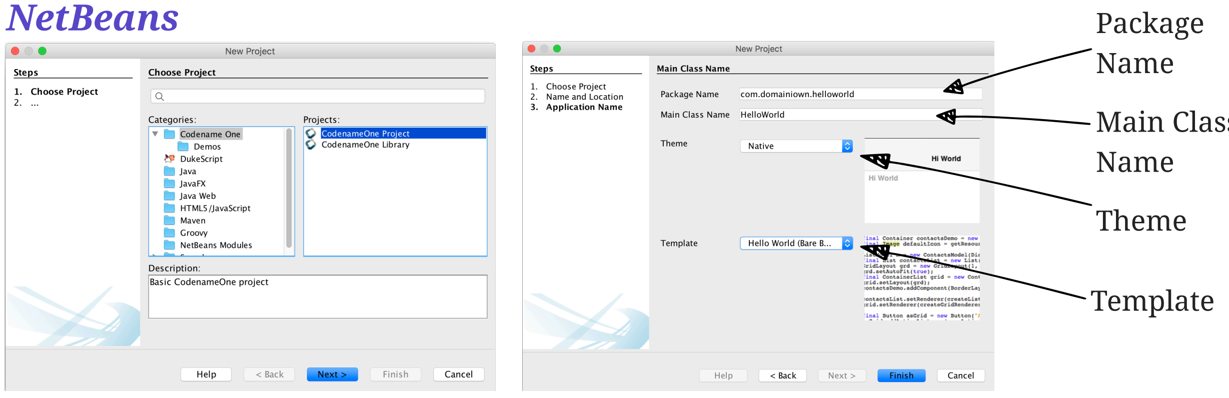
Package Name: com.domainiown.helloworld

Main Class Name: HelloWorld

Theme: Native

Template: Hello World (Bare Bones)

NetBeans


New Project

Steps
1. Choose Project
2. ...

Choose Project

Categories: Codename One, Demos, DukeScript, Java, JavaFX, Java Web, HTML5/JavaScript, Maven, Groovy, NetBeans Modules

Projects: CodenameOne Project, CodenameOne Library

Description: Basic CodenameOne project

New Project

Steps
1. Choose Project
2. Name and Location
3. Application Name

Main Class Name

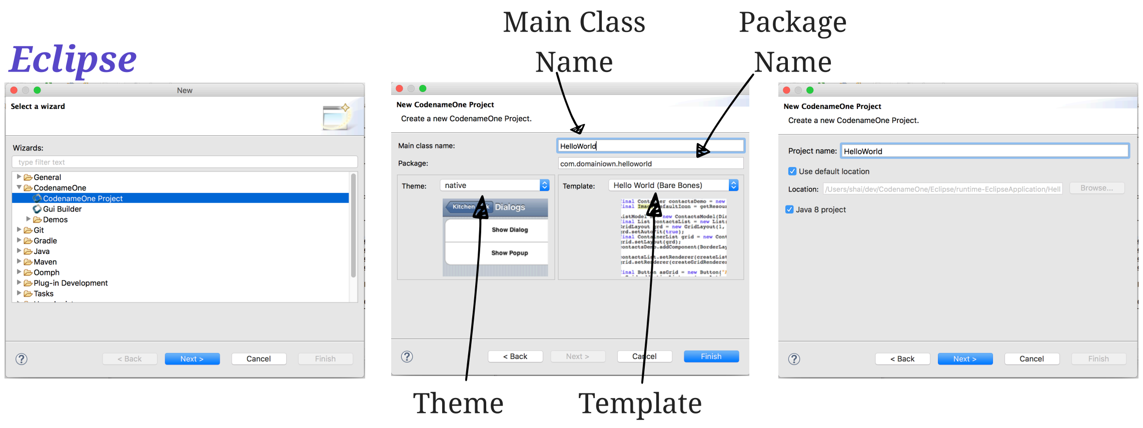
Package Name: com.domainiown.helloworld

Main Class Name: HelloWorld

Theme: Native

Template: Hello World (Bare B...

Eclipse


New

Select a wizard

Wizards:

General, CodenameOne, CodenameOne Project, Gui Builder, Demos, Git, Gradle, Java, Maven, Oomph, Plug-in Development, Tasks

New CodenameOne Project

Create a new CodenameOne Project.

Main class name: HelloWorld

Package: com.domainiown.helloworld

Theme: native

Template: Hello World (Bare Bones)

New CodenameOne Project

Create a new CodenameOne Project.

Project name: HelloWorld

Use default location

Location: /Users/shai/dev/CodenameOne/Eclipse/runtime-EclipseApplication/Hell

Java 8 project

Figure 1. 1. The New App Wizard

Picking a Package Name

Apple, Google and Microsoft identify applications based on their package names. If you use a domain that you don't own it's possible that someone else will use that domain and collide with you. In fact some developers left the default com.mycompany domain in place all the way into production in some cases.

This can cause difficulties when submitting to Apple, Google or Microsoft. Submitting to one of them is no guarantee of success when submitting to another.

To come up with the right package name use a reverse domain notation. So if my website is goodstuff.co.uk my package name should start with uk.co.goodstuff . I highly recommend the following guidelines for package names:

- **Lower Case** – some OS's are case sensitive and handling a mistake in case is painful. The Java convention is lower case and I would recommend sticking to that although it isn't a requirement
- **Avoid Dash and Underscore** – You can't use a dash character (-) for a package name in Java. Underscore (_) doesn't work for iOS. If you want more than one word just use a deeper package e.g.: com.mydomain.deeper.meaningful.name
- **Obey Java Rules** – A package name can't start with a number so you can't use com.mydomain.1sler. You should avoid using Java keywords like this, if etc.
- **Avoid Top Level** – instead of using uk.co.goodstuff use uk.co.goodstuff.myapp. That would allow you to have more than one app on a domain

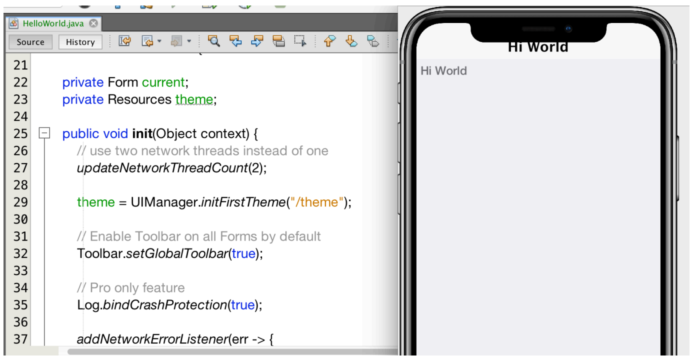
Running

We can run the HelloWorld application by pressing the Play or Run button in the IDE for NetBeans or IntelliJ. In Eclipse we first need to select the simulator .launch file and then press run. When we do that the Codename One simulator launches. You can use the menu of the simulator to control and inspect details related to the device. You can rotate it, determine it's location in the world, monitor networking calls etc.

With the Skins menu you can download device skins to see how your app will look on different devices.

Some skins are bigger than the screen size, uncheck the Scrollable flag in the Simulator menu to handle them more effectively

Debug works just like Run by pressing the IDE's debug button. It allows us to launch the simulator in debug mode where we can set breakpoints, inspect variables etc.



```
    private Form current;
    private Resources theme;

    public void init(Object context) {
        // use two network threads instead of one
        updateNetworkThreadCount(2);

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

        // Enable Toolbar on all Forms by default
        Toolbar.setGlobalToolbar(true);

        // Pro only feature
        Log.bindCrashProtection(true);

        addNetworkErrorListener(err -> {
```

Figure 1. 2. HelloWorld Running on the Simulator with an iPhone X Skin

Simulator vs. Emulator

Codename One ships with a simulator similarly to the iOS toolchain which also has a simulator. Android ships with an emulator. Emulators go the extra mile. They create a virtual machine that's compatible with the device CPU and then boot the full mobile OS within that environment. This provides an accurate runtime environment but is **painfully slow**.

Simulators rely on the fact that OS's are similar and so they leave the low level details in place and just map the API behavior. Since Codename One relies on Java it can start simulating on top of the virtual machine on the desktop. That provides several advantages including fast development cycles and full support for all the development tools/debuggers you can use on the desktop.

Emulators make sense for developers who want to build OS level services e.g. screensavers or low level services. Standard applications are better served by simulators.

The Source Code

After clicking finish in the new project wizard we have a HelloWorld project with a few default settings. I'll break the class down to small pieces and explain each piece starting with the enclosing class:

Listing 1. 1. HelloWorld Class

```
public class HelloWorld {
    private Form current;

    private Resources theme;

    // ... class methods ...
}
```

This is the main class, it's the entry point to the app, notice it doesn't have a main method but rather callback which we will discuss soon

Forms are the "top level" UI element in Codename One. Only one Form is shown at a time and everything you see on the screen is a child of that Form

Every app has a theme, it determines how everything within the application looks e.g. colors, fonts etc.

Next let's discuss the first lifecycle method init(Object). I discuss the lifecycle in depth in the Application Lifecycle Sidebar (page 16).

Listing 1. 2. HelloWorld init(Object)

```
public void init (Object context) {
   updateNetworkThreadCount(2);

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

   Toolbar.setGlobalToolbar(true);

   Log.bindCrashProtection(true);

   addNetworkErrorListener (err -> {

      err.consume();

      if(err.getError() != null){
         Log.e(err.getError());
      }

      Log.sendLogAsync();


      Dialog.show("Connection Error",
         "There was a networking error in the connection to " +
         err.getConnectionRequest().getUrl(), "OK", null);
   });
}
```

init is the first of the four lifecycle methods. It's responsible for initialization of variables and values

By default Codename One has one thread that performs all the networking, we set the default to two which gives better performance

The theme determines the appearance of the application. We'll discuss this in the next chapter

This enables the Toolbar API by default, it allows finer control over the title bar area

Crash protection automatically sends device crash logs through the cloud

In case of a network error the code in this block would run, you can customize it to handle networking errors effectively. consume() swallows the event so it doesn't trigger other alerts, it generally means "we got this"

Not all errors include an exception, if we have an exception we can log it with this code

This will email the log from the device to you if you have a pro subscription

This shows an error dialog to the user, in production you might want to remove that code init(Object) works as a constructor to some degree. We recommend avoiding the constructor for the main class and placing logic in the init method instead. This isn't crucial but we recommend it since the constructor might happen too early in the application lifecycle.

In a cold start init(Object) is invoked followed by the start() method. However, start() can be invoked more than once if an app is minimized and restored, see the sidebar Application Lifecycle (page 16):

Listing 1. 3. HelloWorld start()

```java
public void start() {
    if(current != null) {
        current.show();
        return;
    }
    Form hi = new Form("Hi World", BoxLayout.y());
    hi.add(new Label("Hi World"));
    hi.show();
}
```

If the app was minimized we usually don't want to do much, just show the last Form of the application

current is a Form which is the top most visual element. We can only have one Form showing and we enforce that by using the show() method

We create a new simple Form instance. It has the title "Hello World" and arranges elements vertically (on the Y axis)

We add another Label below the title, see figure 1.3 (page 15). I discuss component hierarchy later in section 2.2 (page 37)

The show() method places the Form on the screen. Only one Form can be shown at a time

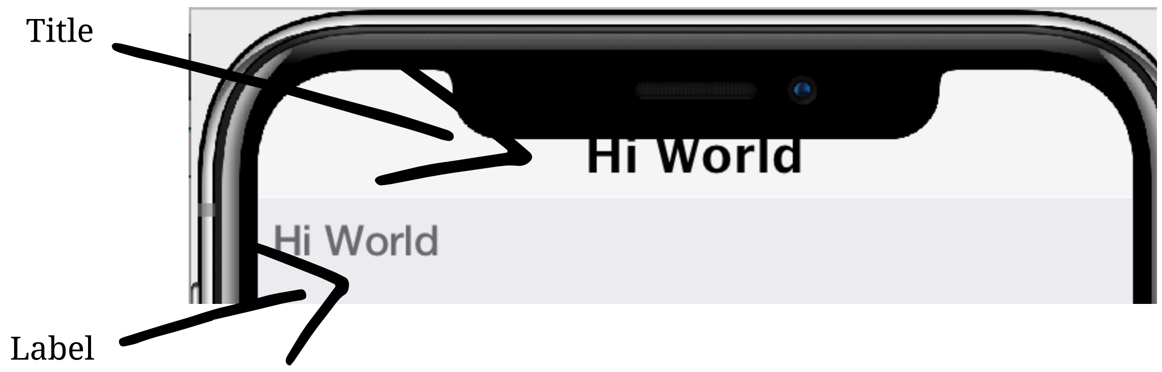


Figure 1. 3. Title and Label in the UI

There are some complex ideas within this short snippet which I'll address later in this chapter when talking about layout. The gist of it is that we create and show a Form. Form is the top level UI element, it takes over the whole screen. We can add UI elements to that Form object, in this case the Label. We use the BoxLayout to arrange the elements within the Form from top to the bottom vertically.

1

Application Lifecycle

A few years ago Romain Guy (a senior Google Android engineer) was on stage at the Google IO conference. He asked for a show of hands of people who understand the Activity lifecycle (Activity is similar to a Codename One main class). He then proceeded to jokingly call the audience members who lifted their hands “liars” claiming that after all his years in Google he still doesn’t understand it…

Lifecycle seems simple on the surface but hides a lot of nuance. Android’s lifecycle is ridiculously complex. Codename One tries to simplify this and also make it portable. Sometimes complexity leaks out and the nuances can be difficult to deal with.

Simply explained an application has three states:

- ***Foreground*** – it’s running and in the foreground which means the user can physically interact with the app
- ***Suspended*** – the app isn’t in the foreground, it’s either paused or has a background process running
- ***Not Running*** – the app was never launched, was killed or crashed

The lifecycle is the process of transitioning between these 3 states and the callbacks invoked when such a transition occurs. The first time we launch the app we start from a “Cold Start” (Not Running State) but on subsequent launches the app is usually started from the "Warm Start" (Suspended State).

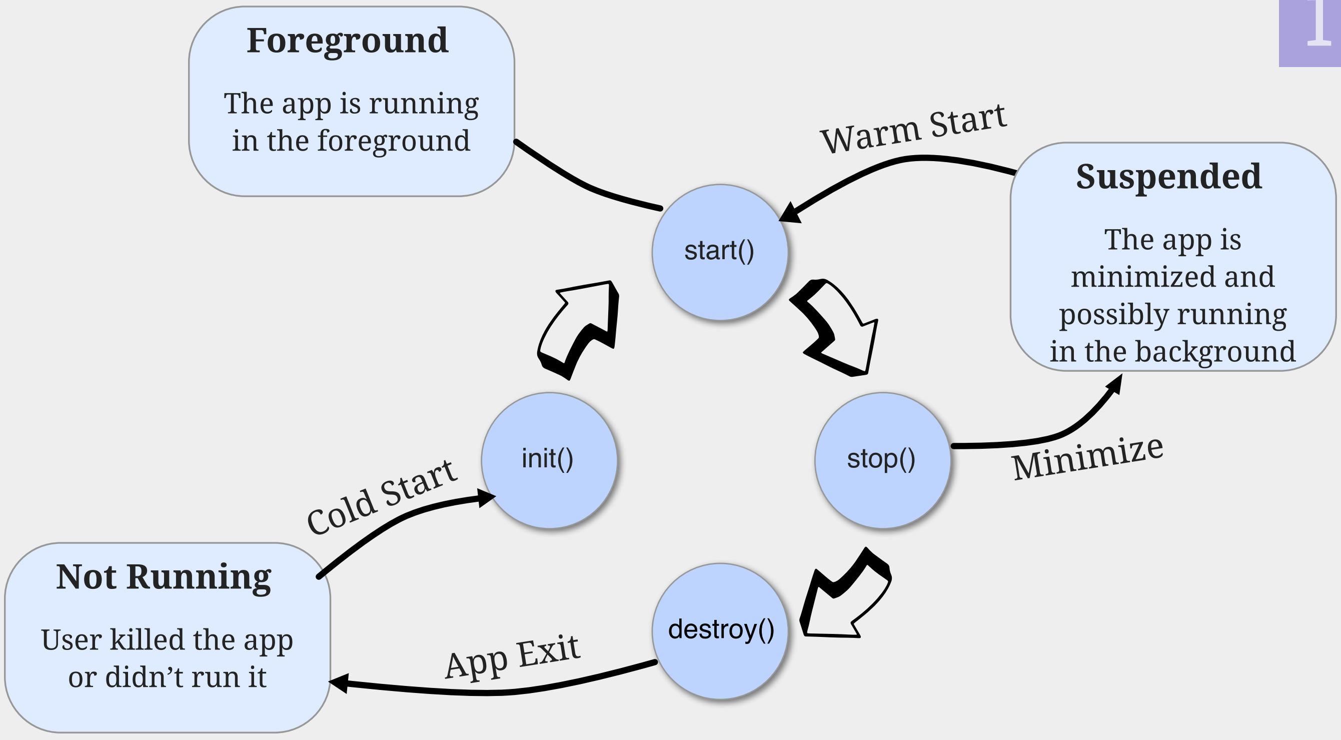

Foreground
The app is running in the foreground
Warm Start
Suspended
The app is minimized and possibly running in the background
start()
init()
stop()
Minimize
Cold Start
Not Running
User killed the app or didn't run it
App Exit
destroy()

Figure 1. 4. Codename One Application Lifecycle

Codename One has four standard callback methods in the lifecycle API:

- init(Object) – is invoked when the app is first launched from a *Not Running* state
- start() – is invoked for two separate cases. After start() is finished the app transitions to the *Foreground* state.
  - Following init(Object) in case of a cold start. Cold start refers to starting the app from a *Not Running* state.
  - When the app is restored from *Suspended* state. In this case init(Object) isn't invoked
- stop() – is invoked when the app is minimized e.g. when switching to a different app. After stop() is finished the app transitions to the *Suspended* state.
- destroy() – is invoked when the app is destroyed e.g. killed by a user in the task manager. After destroy() is finished the app is no longer running hence it's in the *Not Running* state.

destroy() is optional there is no guarantee that it would be invoked. It should be used only as a last resort

Now that we have a general sense of the lifecycle lets look at the last two lifecycle methods:

Listing 1. 4. HelloWorld stop() and destroy()

```
public  void  stop()  {
     current  = getCurrentForm();
     if ( current  instanceof  Dialog )  {
          ((Dialog) current ).dispose ();
          current  =  getCurrentForm();
     }
}
```

stop() is invoked when the app is minimized or a different app is opened

As the app is stopped we save the current Form so we can restore it back in start() if the app is restored

Dialog is a bit of a special case restoring a Dialog might block the proper flow of application execution so we dispose them and then get the parent Form

```
public  void  destroy ()  {
}
```

destroy() is a very special case. Under normal circumstances you shouldn't write code in destroy(). stop() should work for most cases

That's it. Hopefully you have a general sense of the code. It's time to run on the device.

1.2.2 *Run on Device*

Now that we have a HelloWorld and a basic understanding of the lifecycle lets discuss building apps for devices. I'll only discuss Android and iOS for simplicity.

While Codename One supports Windows and a few other platforms the focus of this book is on Android/iOS to keep things manageable. Windows is a more significant player in the tablet market which isn't as applicable for this app

Signing

All of the modern mobile platforms require signed applications but they all take radically different approaches when implementing it.

Signing is a process that marks your final application for the device with a special value. This value (signature) is a value that only you can generate based on the content of the application and your certificate. Effectively it guarantees the app came from you. This blocks a 3rd party from signing their apps and posing as you to the appstore or to the user. It's a crucial security layer.

A certificate is the tool we use for signing. Think of it as a mathematical rubber stamp that generates a different value each time. Unlike a rubber stamp a signature can't be forged!

Signing on Android

Android uses a self signed certificate approach. You can just generate a certificate by describing who you are and picking a password!

Anyone can do that. However, once a certificate is generated it can’t be replaced…

If you lose an Android certificate it can’t be restored and you won’t be able to update your app!

If this wasn’t the case someone else could potentially push an “upgrade” to your app. Once an app is submitted with a certificate to Google Play this app can’t be updated with any other certificate.

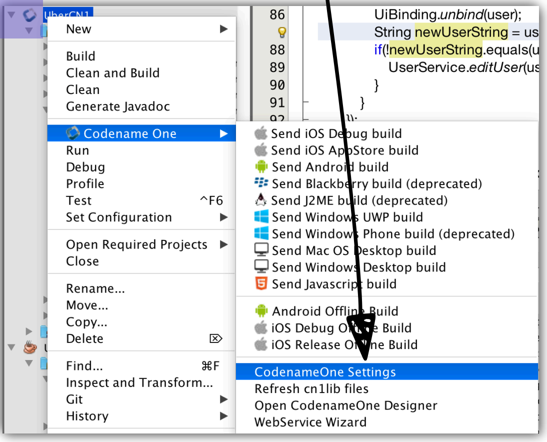
With that in mind generating an Android certificate is trivial.

The following chart illustrates a process that’s identical on all IDE’s

Your certificate will generate into the file `Keychain.ks` ***in your home directory***

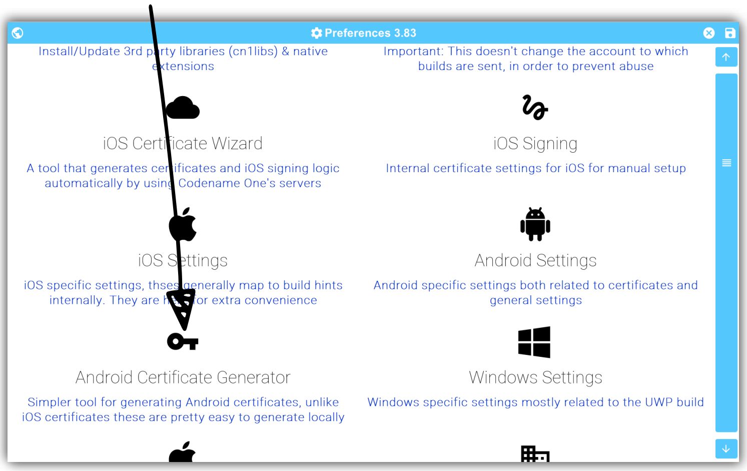
Make sure to back that up and the password as losing these can have dire consequences

1 Right click the project and select “Codename One Settings”



```
            UIBinding.unbind(user);
            String newUserString = us
            if(!newUserString.equals(u
                UserService.editUser(us
            }
        }
```

Click “Android Cetificate Generator”



Install/Update 3rd party libraries (cn1libs) & native extensions

Important: This doesn't change the account to which builds are sent, in order to prevent abuse

A tool that generates certificates and iOS signing logic automatically by using Codename One's servers

Internal certificate settings for iOS for manual setup

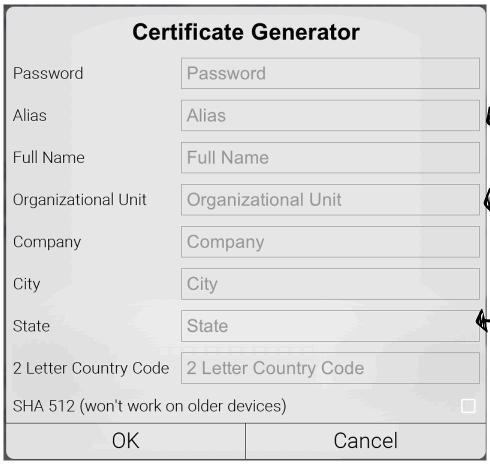
iOS specific settings, these generally map to build hints internally. They are here for extra convenience

Android specific settings both related to certificates and general settings

Simpler tool for generating Android certificates, unlike iOS certificates these are pretty easy to generate locally

Windows specific settings mostly related to the UWP build

Don’t forget the password



Password	Password
Alias	Alias
Full Name	Full Name
Organizational Unit	Organizational Unit
Company	Company
City	City
State	State
2 Letter Country Code	2 Letter Country Code
SHA 512 (won't work on older devices)	☐

Alias is a simple ID for the certificate e.g. codenameone

The other details will be visible to the users of the app when they inspect your apps signature

Checking this will decrease the likelyhood of anyone forging this certificate in the forseeable future. Notice that the current likelyhood is **very** low. The new 512bit certificates only work on Android 4.1 or newer

Press OK when you are

Figure 1. 5. Process of Certificate Generation for Android

Should I Use a Different Certificate for Each App?

In theory yes. In practice it’s a pain… Keeping multiple certificates and managing them is a pain so we often just use one.

The drawback of this approach occurs when you are building an app for someone else or want to sell the app. Giving away your certificate is akin to giving away your house keys. So it makes sense to have separate certificates for each app.

Signing and Provisioning iOS

Code signing for iOS relies on Apple as the certificate authority. This is something that doesn't exist on Android. iOS also requires provisioning as part of the certificate process and completely separates the process for development/release.

But first let's start with the good news:

- Losing an iOS certificate is no big deal - in fact we revoke them often with no impact on shipping apps
- Codename One has a wizard that hides most of the pain related to iOS signing

In iOS Apple issues the certificates for your applications. That way the certificate is trusted by Apple and is assigned to your Apple iOS developer account. There is one important caveat: You need an iOS Developer Account and Apple charges a 99USD Annual fee for that.

> The 99USD price and requirement have been around since the introduction of the iOS developer program for roughly 10 years at the time of this writing. It might change at some point though

Apple also requires a "provisioning profile" which is a special file bound to your certificate and app. This file describes some details about the app to the iOS installation process. One of the details it includes during development is the list of permitted devices.

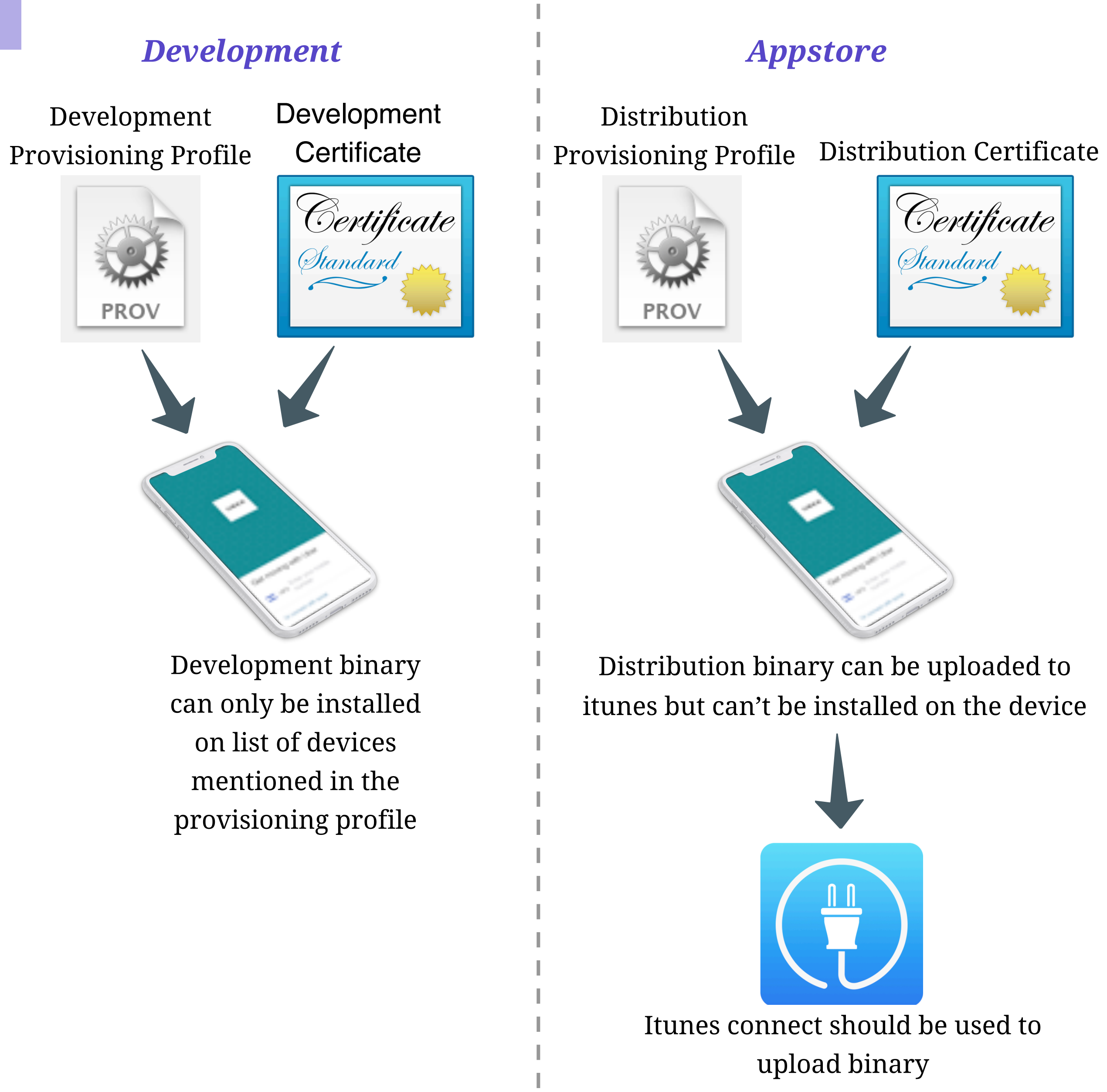

Development
Development Provisioning Profile
Development Certificate
PROV
Certificate Standard
Development binary can only be installed on list of devices mentioned in the provisioning profile
Appstore
Distribution Provisioning Profile
Distribution Certificate
PROV
Certificate Standard
Distribution binary can be uploaded to itunes but can't be installed on the device
Itunes connect should be used to upload binary

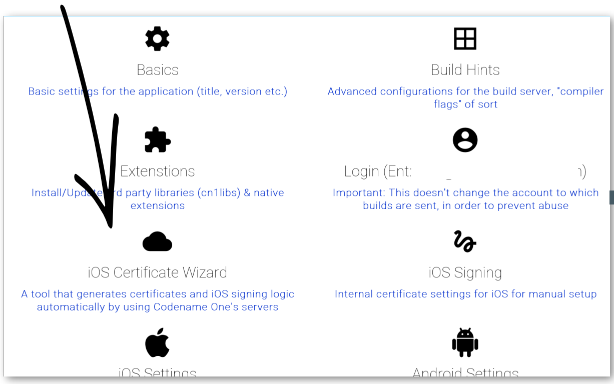
Figure 1. 6. The Four Files Required for iOS Signing and Provisioning

We need 4 files for signing. Two certificates and two provisioning profiles:

1. Production - The production certificate/provisioning pair is used for builds that are uploaded to iTunes
2. Development - The development certificate/provisioning is used to install on your development devices

The certificate wizard automatically creates these 4 files and configures them for you.

1 In Codename One Settings click iOS Certificate Wizard


Basics

Basic settings for the application (title, version etc.)

Build Hints

Advanced configurations for the build server, "compiler flags" of sort

Extenstions

Install/Update 3rd party libraries (cn1libs) & native extensions

Login (Ent: _)

Important: This doesn't change the account to which builds are sent, in order to prevent abuse

iOS Certificate Wizard

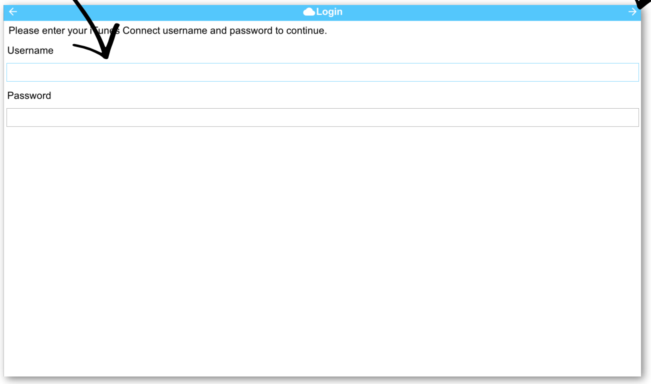
A tool that generates certificates and iOS signing logic automatically by using Codename One's servers

iOS Signing

Internal certificate settings for iOS for manual setup

2 These are the Apple iOS developer program email and password pair. Not the Codename One password!

Next to proceed



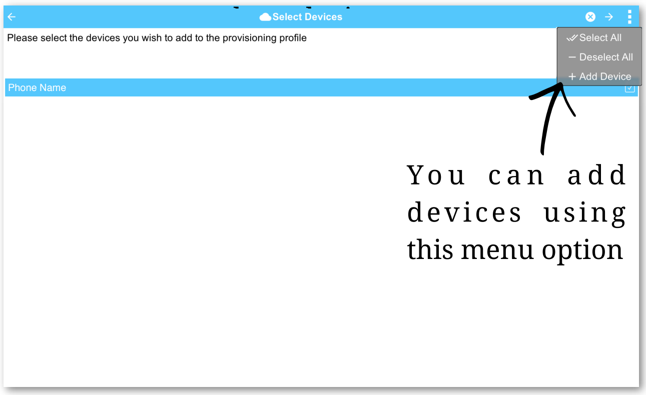
Please enter your iTunes Connect username and password to continue.

Username

Password

Figure 1. 7. Using the iOS Certificate Wizard Steps 1 and 2

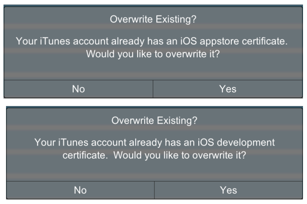
3 Here we have the list of devices that you can add to the provisioning profile. You can install



Please select the devices you wish to add to the provisioning profile

You can add devices using this menu option

4 If you have an existing certificate you will be offered to revoke it


Overwrite Existing?

Your iTunes account already has an iOS appstore certificate. Would you like to overwrite it?

No | Yes

Overwrite Existing?

Your iTunes account already has an iOS development certificate. Would you like to overwrite it?

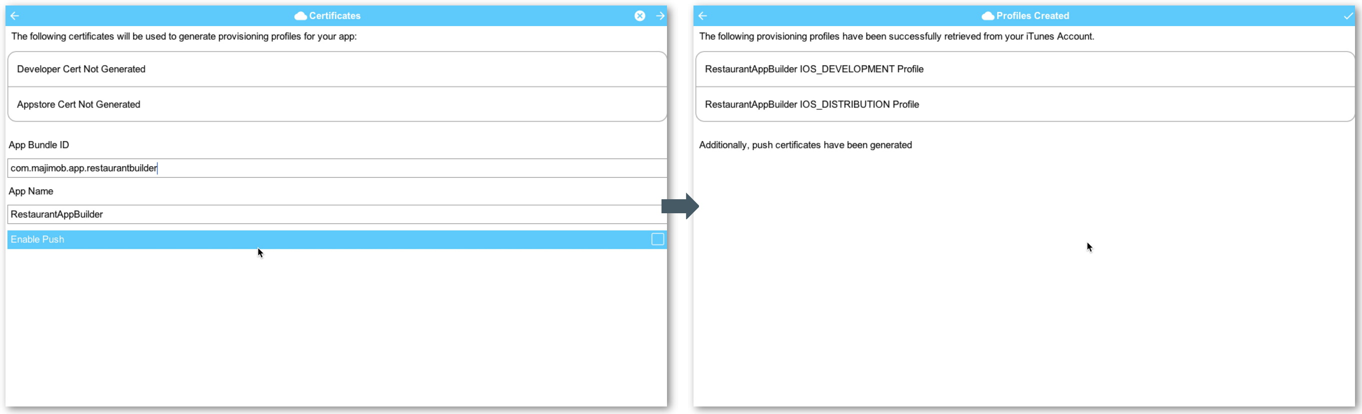
No | Yes

If your existing certificate is fine, you shouldn't revoke just share the single P12 file between projects

Figure 1. 8. Using the iOS Certificate Wizard Steps 3 and 4

5 You are shown the details of the files that should be generates

6 The final form shows a summary of what was performed by the wizard


Certificates

The following certificates will be used to generate provisioning profiles for your app:

Developer Cert Not Generated

Appstore Cert Not Generated

App Bundle ID: com.majimob.app.restaurantbuilder

App Name: RestaurantAppBuilder

Enable Push

Profiles Created

The following provisioning profiles have been successfully retrieved from your iTunes Account.

RestaurantAppBuilder IOS_DEVELOPMENT Profile

RestaurantAppBuilder IOS_DISTRIBUTION Profile

Additionally, push certificates have been generated

Figure 1. 9. Using the iOS Certificate Wizard Steps 5 and 6

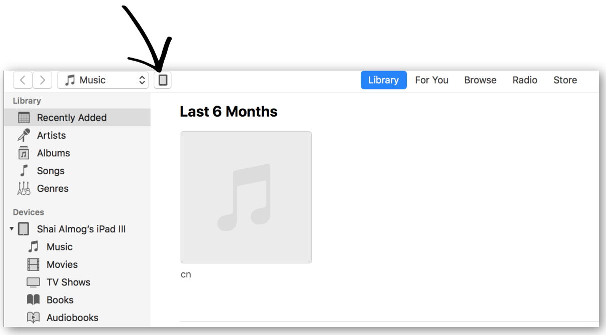
If you have more than one project you should use the same iOS P12 certificate files in all the projects and just regenerate the provisioning. In this situation the certificate wizard asks you if you want to revoke the existing certificate which you shouldn't revoke in such a case. You can update the provisioning profile in Apple's iOS developer website.

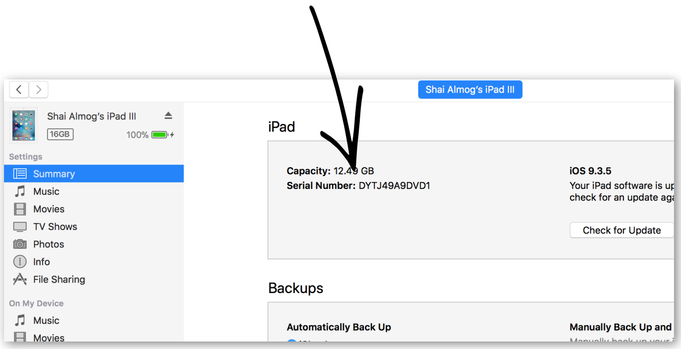
One important aspect of provisioning on iOS is the device list in the provisioning step. Apple only allows you to install the app on 100 devices during development. This blocks developers from skipping the appstore altogether. It's important you list the correct UDID for the device in the list otherwise install will fail.

There are several apps and tools that offer the UDID of the device, they aren't necessarily reliable and might give a fake number!

1 To get the UDID connect your iDevice to your computer and launch iTunes. Then click on the device icon

2 Click the serial number of the device

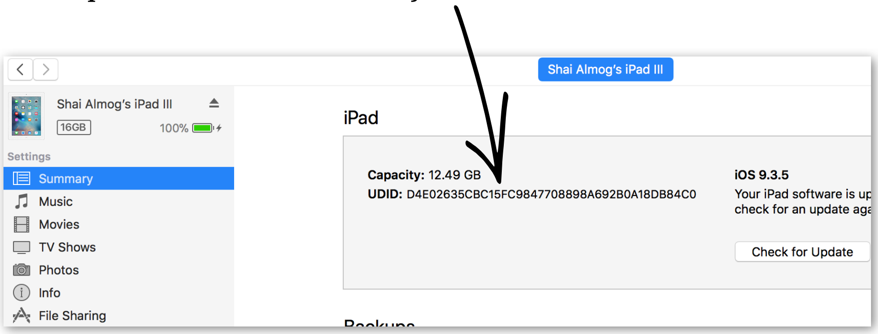




Capacity: 12.49 GB
Serial Number: DYTJ49A9DVD1

Your iPad software is up
check for an update aga

3 The serial number turns to the UDID. Notice that this is in the same UI view. The serial number updates to the UDID when you click on it



Capacity: 12.49 GB
UDID: D4E02635CBC15FC9847708898A692B0A18DB84C0

Your iPad software is up
check for an update aga

Figure 1. 10. Get the UDID of a Device

You can right click the UDID and select copy to copy it

The simplest and most reliable process for getting a UDID is via itunes. I've used other approaches in the past that worked but this approach is guaranteed.

Ad hoc provisioning allows 1000 beta testers for your application but it's a more complex process that I won't discuss here

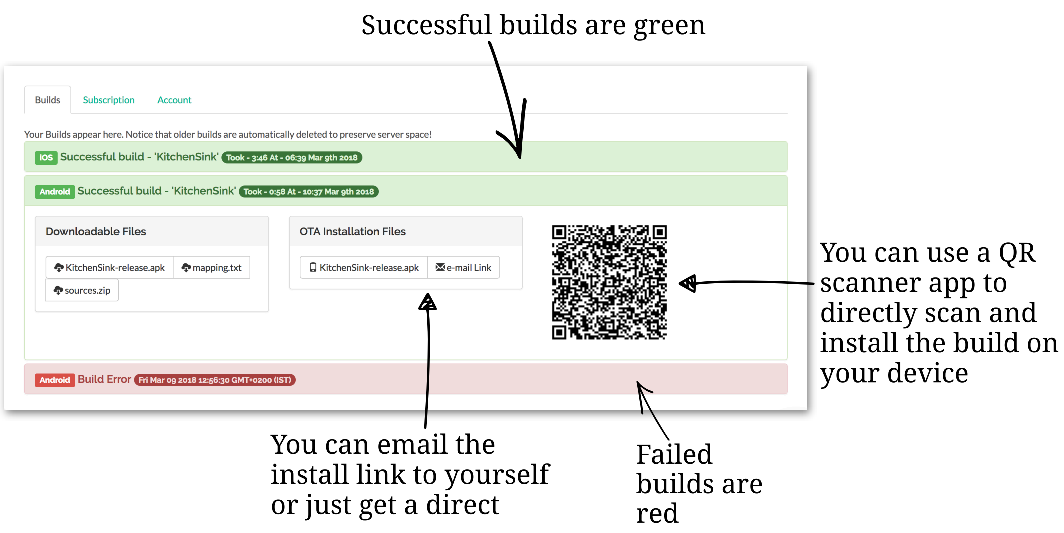
Build and Install

Before we continue with the build we should sign up at www.codenameone.com/build-server.html where you can soon follow the progress of your builds. I discuss this further in section 1.3 (page 26). You need a Codename One account in order to build for the device.

Now that we have certificates the process of device builds is literally a right click away for both OS's. We can right click the project and select Codename One → Send iOS Debug Build or Codename One → Send Android Build.

The first time you send a build you will be prompted for the email and password you provided when signing up for Codename One

Once you send a build you should see the results in the build server page:


Successful builds are green
You can use a QR scanner app to directly scan and install the build on your device
You can email the install link to yourself or just get a direct
Failed builds are red
Builds
Subscription
Account
Downloadable Files
KitchenSink-release.apk
mapping.txt
sources.zip
OTA Installation Files
KitchenSink-release.apk
e-mail Link

Your Builds appear here. Notice that older builds are automatically deleted to preserve server space!

iOS Successful build - 'KitchenSink' Took - 3:46 At - 06:39 Mar 9th 2018

Android Successful build - 'KitchenSink' Took - 0:58 At - 10:37 Mar 9th 2018

Android Build Error Fri Mar 09 2018 12:56:30 GMT+0200 (IST)

Figure 1. 11. Build Results

On iOS make sure you use Safari when installing, as 3rd party browsers might have issues

Once you go through those steps you should have the `HelloWorld` app running on your device. This process is non-trivial when starting so if you run into difficulties don’t despair and seek help at codenameone.com. Once you go through signing and installation, it becomes easier.

1.3 How Does it Work?

Let’s step back a bit from the `HelloWorld` app and explain what we just did.

As a developer, your view of Codename One is relatively simple:

- You develop your app in Java and the Codename One API
- You debug the app with the Codename One device Simulator
- When you need a native app you can right click the project and select `Send Build for iOS` (or Android, Windows etc.)

That’s it. There is quite a lot more to it but the basic premise is pretty close.

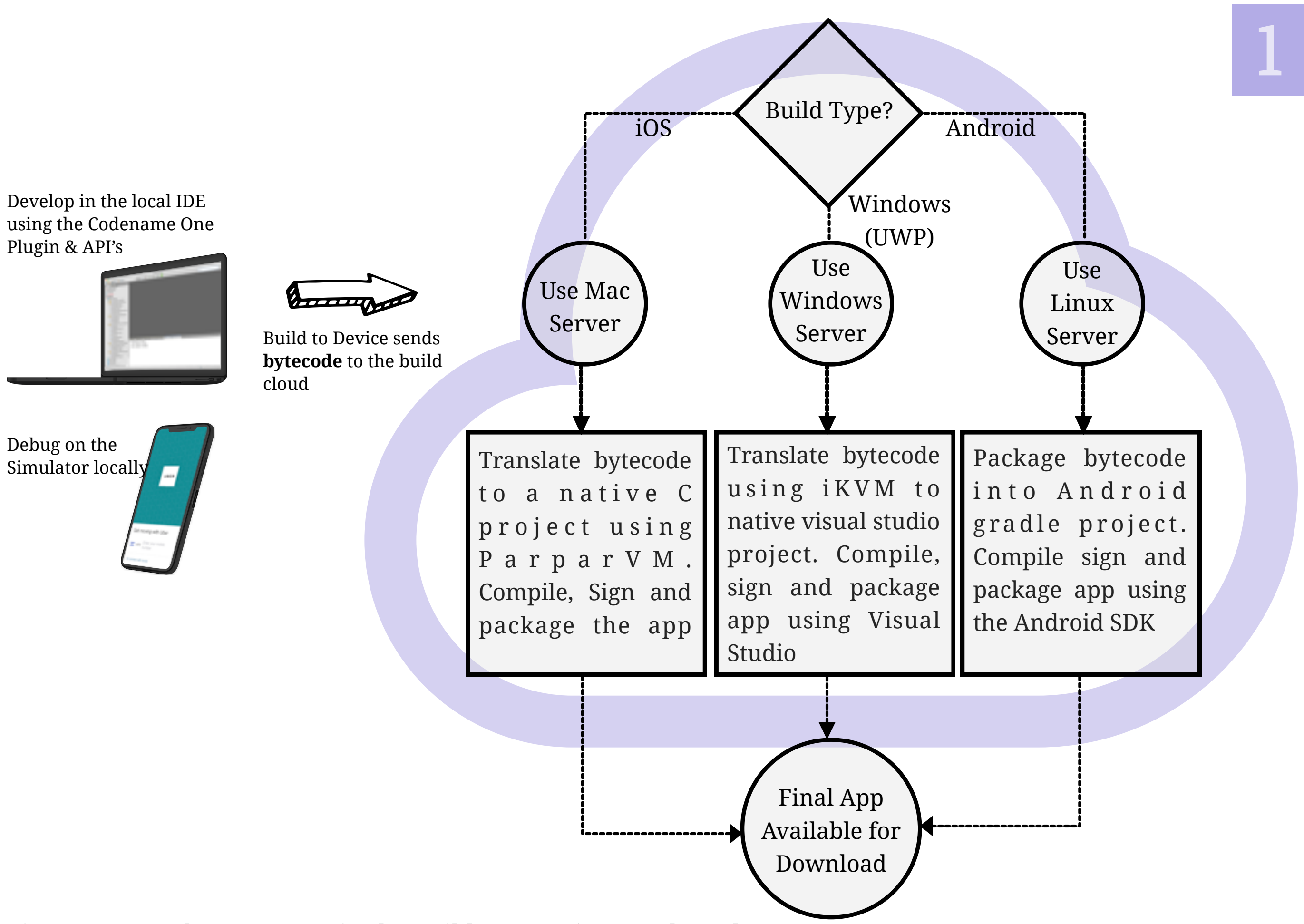

Develop in the local IDE using the Codename One Plugin & API's
Build to Device sends **bytecode** to the build cloud
Debug on the Simulator locally
Build Type?
iOS
Android
Windows (UWP)
Use Mac Server
Use Windows Server
Use Linux Server
Translate bytecode to a native C project using ParparVM. Compile, Sign and package the app
Translate bytecode using iKVM to native visual studio project. Compile, sign and package app using Visual Studio
Package bytecode into Android gradle project. Compile sign and package app using the Android SDK
Final App Available for Download

Figure 1. 12. What Happens in the Build Servers in Broad Strokes

This should give a general sense of the process under the hood. For a more thorough explanation check out Appendix F (page 415).

Mobile is Different

Before we proceed I'd like to explain some universal core concepts of mobile programming that might not be intuitive. These are universal concepts that apply to mobile programming regardless of the tools you are using.

Density (DPI/PPI)

Density is also known as DPI (Dots Per Inch) or PPI (pixels or points per inch). Density is confusing, unintuitive and might collide with common sense. E.g. an iPhone 7 plus has a resolution of 1080x1920 pixels and a PPI of 401 for a 5 inch screen. On the other hand an iPad 4 has 1536x2048 pixels with a PPI of 264 on a 9.7 inch screen… Smaller devices can have higher resolutions!

As the following figure shows, if a Pixel 2 XL had pixels the size of an iPad it would have been twice the size of that iPad. While in reality it's nearly half the height of the iPad!

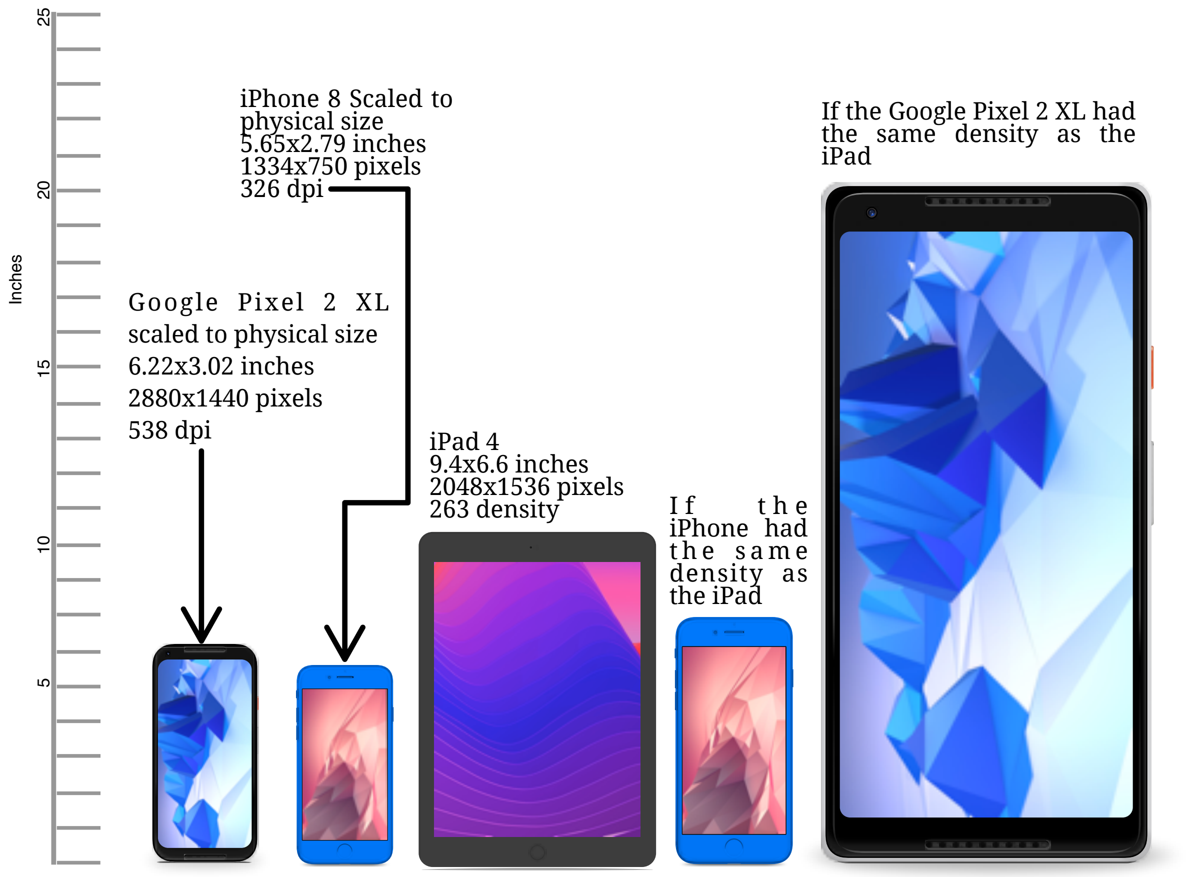

25
20
15
10
5
Inches
iPhone 8 Scaled to physical size
5.65x2.79 inches
1334x750 pixels
326 dpi
Google Pixel 2 XL scaled to physical size
6.22x3.02 inches
2880x1440 pixels
538 dpi
iPad 4
9.4x6.6 inches
2048x1536 pixels
263 density
If the iPhone had the same density as the iPad
If the Google Pixel 2 XL had the same density as the iPad

Figure 1. 13. Device Density vs. Resolution

Differences in density can be extreme. A second generation iPad has 132 PPI, where modern phones have PPI that crosses the 600 mark. Low resolution images on high PPI devices will look either small or pixelated. High resolution images on low PPI devices will look huge, overscaled (artifacts) and will consume too much memory.

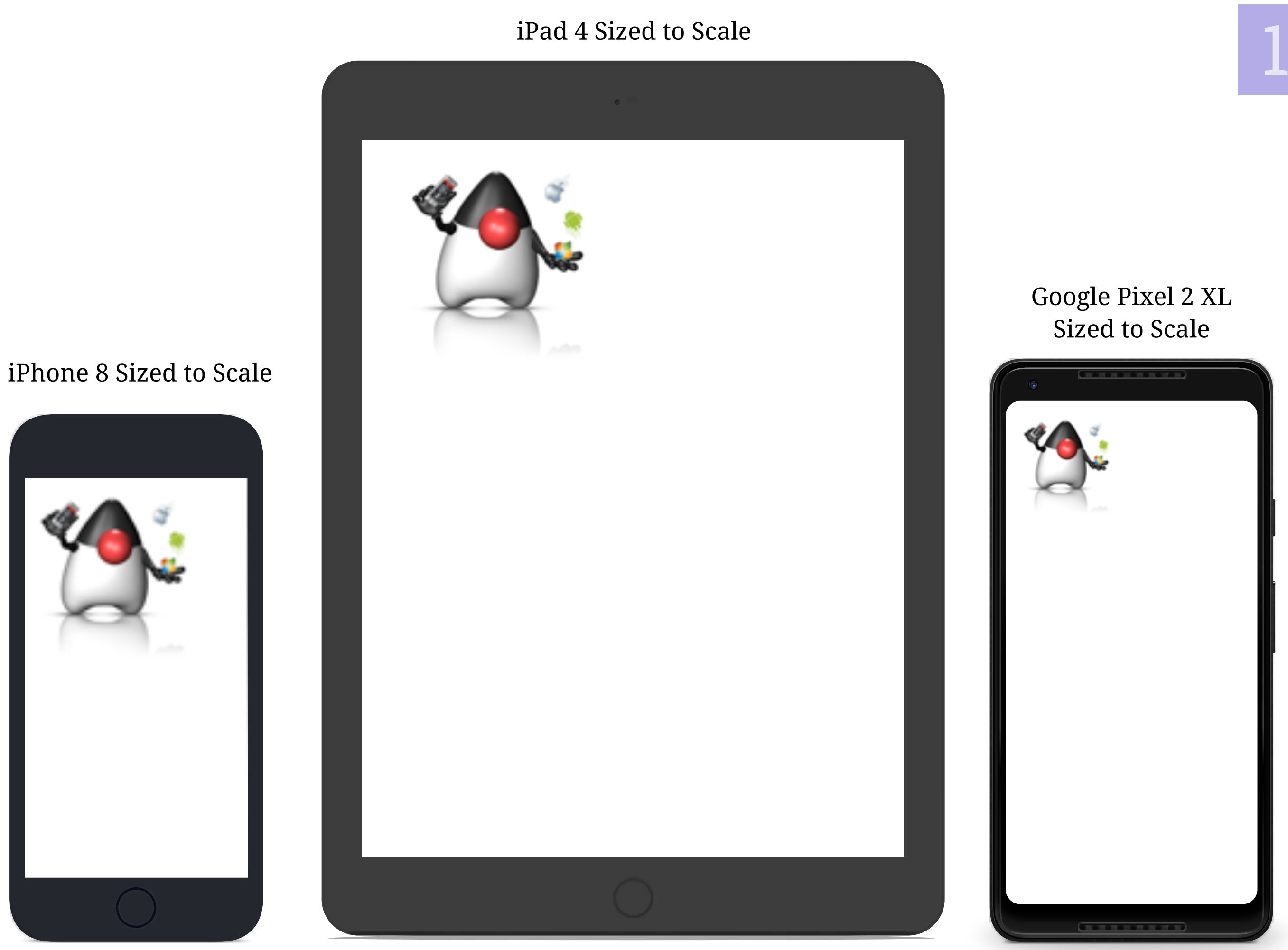


Figure 1. 14. How the Same Image Looks in Different Devices

The exact same image will look different on each device, sometimes to a comical effect. One of the solutions for this problem is multi-images. All OS's support the ability to define different images for various densities. I will discuss multi-images in Chapter 2.

This also highlights the need for working with measurements other than pixels. Codename One supports millimeters (or dips) as a unit of measurement. This is highly convenient and is a better representation of size when dealing with mobile devices.

But there is a bigger conceptual issue involved. We need to build a UI that adapts to the wide differences in form factors. We might have fewer pixels on an iPad but because of its physical size we would expect the app to cram more information into that space so the app won't feel like a blown up phone application. There are multiple strategies to address that but one of the first steps is in the layout managers.

I'll discuss the layout managers in depth in Chapter 2 but the core concept is that they decide where a

UI element is placed based on generic logic. That way the user interface can adapt automatically to the huge variance in display size and density.

Touch Interface

The fact that mobile devices use a touch interface today isn't news… But the implications of that aren't immediately obvious to some developers.

UI elements need to be finger sized and heavily spaced. Otherwise we risk the "fat finger" effect. That means spacing should be in millimeters and not in pixels due to device density.

Scrolling poses another challenge in touch based interfaces. In desktop applications it's very common to nest scrollable items. However, in touch interfaces the scrolling gesture doesn't allow such nuance. Furthermore, scrolling on both the horizontal and vertical axis (side scrolling) can be very inconvenient in touch based interfaces.

Device Fragmentation

Some developers single out this wide range of resolutions and densities as "device fragmentation". While it does contribute to development complexity for the most part it isn't a difficult problem to overcome.

Densities aren't the cause of device fragmentation. Device fragmentation is caused by multiple OS versions with different behaviors. This is very obvious on Android and for the most part relates to the slow rollout of Android vendor versions compared to Googles rollout. E.g. 7 months after the Android 8 (Oreo) release in 2018 it was still available on 1.1% of the devices. The damning statistic is that 12% of the devices in mid 2018 run Android 4.4 Kitkat released in 2013!

This makes QA difficult as the disparity between these versions is pretty big. These numbers will be out of date by the time you read this but the core problem remains. It's hard to get all device manufacturers on the same page so this problem will probably remain in the foreseeable future despite everything.

1.3.2 *Performance*

Besides the obvious need for performance and smooth animation within a mobile app there are a couple of performance related issues that might not be intuitive to new developers: size and power.

App Size

Apps are installed and managed via stores. This poses some restrictions about what an app can do. But it also creates a huge opportunity. Stores manage automatic update and to some degree the marketing/monetization of the app.

A good mobile app is updated once a month and sometimes even once a week. Since the app downloads automatically from the store this can be a huge benefit:

- Existing users are reminded of the app and get new features instantly
- New users notice the app featured on a "what's new" list

If an app is big it might not update over a cellular network connection. Google and Apple have restrictions on automatic updates over cellular networks to preserve battery life and data plans. A large app might negatively impact users perception of the app and trigger uninstalls e.g. when a phone is low on available space.

Power Drain

Desktop developers rarely think about power usage within their apps. In mobile development this is a crucial concept. Modern device OS's have tools that highlight misbehaving applications and this can lead to bad reviews.

Code that loops forever while waiting for input will block the CPU from sleeping and slowly drain the battery.

Worse. Mobile OS's kill applications that drain the battery. If the app is draining the battery and is minimized (e.g. during an incoming call) the app could be killed. This will impact app performance and usability.

Sandbox and Permissions

Apps installed on the device are "sandboxed" to a specific area so they won't harm the device or its functionality. The filesystem of mobile applications is restricted so one application can't access the files of another application. Things that most developers take for granted on the desktop such as a "file picker" or accessing the image folder don't work on devices!

This means that when your application works on a file it belongs only to your application. In order to share the file with a different application you need to ask the operating system to do that for you.

Furthermore, some features require a "permission" prompt and in some cases require special flags in system files. Apps need to request permission to use sensitive capabilities e.g. Camera, Contacts etc. Historically Android developers just declared required permissions for an app and the user was prompted with permissions during install. Android 6 adopted the approach used by iOS of prompting the user for permission when accessing a feature.

This means that in runtime a user might revoke a permission. A good example in the case of an Uber app is the location permission. If a user revokes that permission the app might lose its location.

Ubers Permission Controversy

While working on this book I was surprised that the Uber app didn't include some common functionality in Android applications (namely SMS intercept). As I researched this it seems that in the past the Uber app used to have a **huge** set of permissions. This raised privacy concerns among power users and produced backlash of users calling for a ban.

It seems that Uber decided to take permissions seriously. They don't ask for permissions even if it comes at the expense of reduced functionality. I think that decision was made prior to Android 6 which gives end users more control over permissions and Uber should probably revisit this policy.

I think this is an important thing to keep in mind when thinking about permissions. It might be advantageous to avoid a feature if it has problematic permissions.

1.4 *Summary*

In this chapter, we learned:

- How we can correctly define package names so vendors such as Apple and Google identify/update our app correctly
- How we should use the mobile application lifecycle to handle app state changes
- How various mobile device screen sizes impact your UI, and how to work around that to make your app work across PPI limitations
- How to understand the mobile development landscape and differentiate between it and desktop programming
- How to sign/provision a mobile app so we can build and run on a mobile device

I barely scratched the surface of Codename One in this chapter… It's a huge framework. I will go into more details in the next chapter.

Further Reading

Codename One has over a decade’s worth of code and knowledge. I made an effort to make this book “all inclusive”, but there are still limits to this medium. New platforms and tools are always a challenge, that’s why we try to help with any question. So please engage with the online community if something doesn’t work…

- Codename One Developer Guide – www.codenameone.com/manual/
- JavaDocs – www.codenameone.com/javadoc/
- Technical Support – stackoverflow.com/tags/codenameone or www.codenameone.com/discussion-forum.html
- Short Video Tutorials – www.codenameone.com/how-do-i.html
- Online course – codenameone.teachable.com/

Core Concepts 2

This chapter covers:

- Laying out a UI, component hierarchy and nesting
- Using the Themes in the Codename One Designer tool to style an app
- Events processing, the event dispatch thread and Storage

Now that we have a general sense of how a hello world works and some broad stroke overview of mobile development, lets see how this all fits into the concepts of Codename One. We'll accomplish that by building a small Todo list app. While we do that I'll try to explain how everything works together in the grand scheme of things…

I'll try to keep this short so we will have all the tools we need to build the Uber app client by the end of the chapter.

The Todo App 2.1

This is what we should end up with before this chapter is finished:

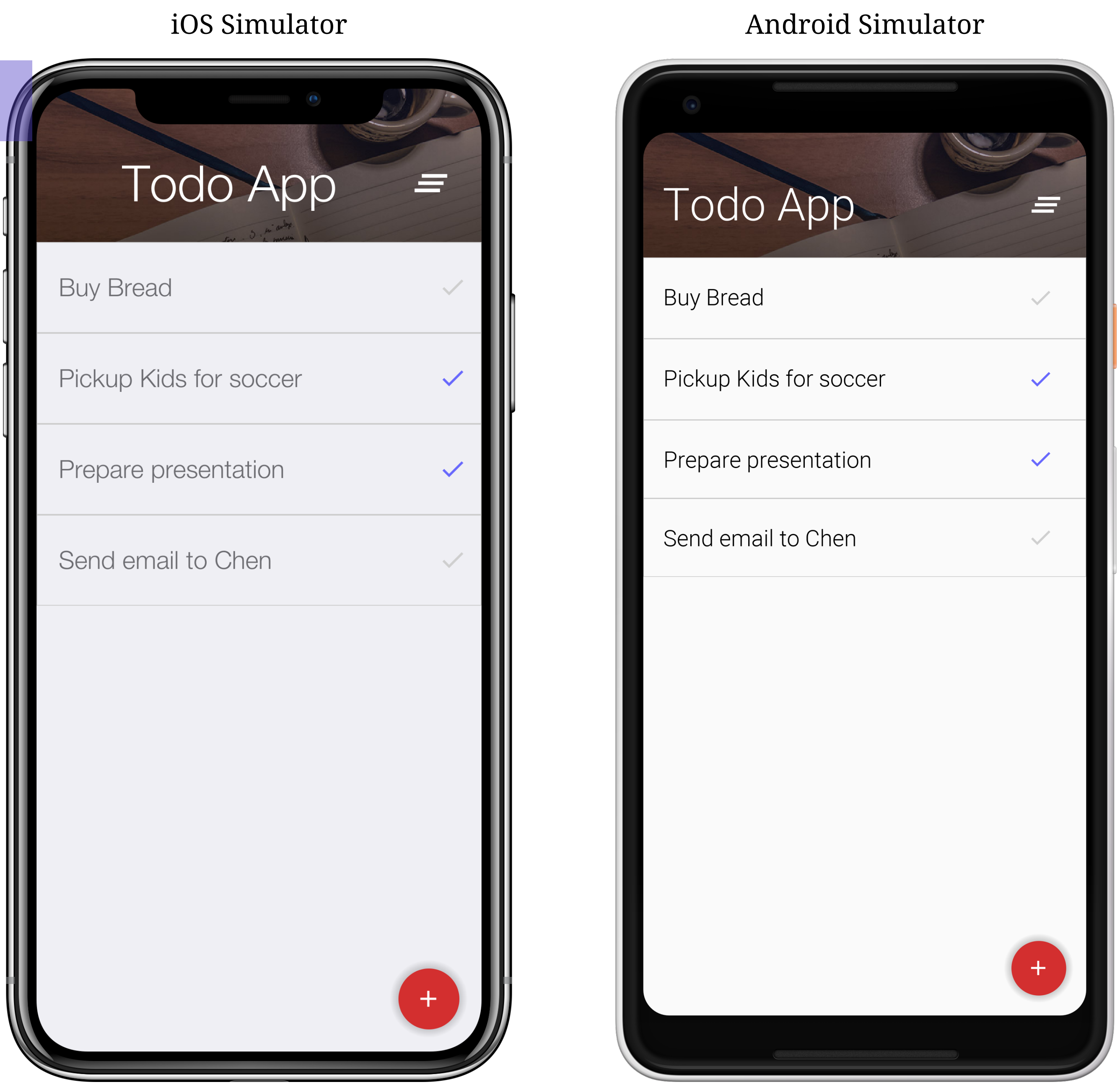
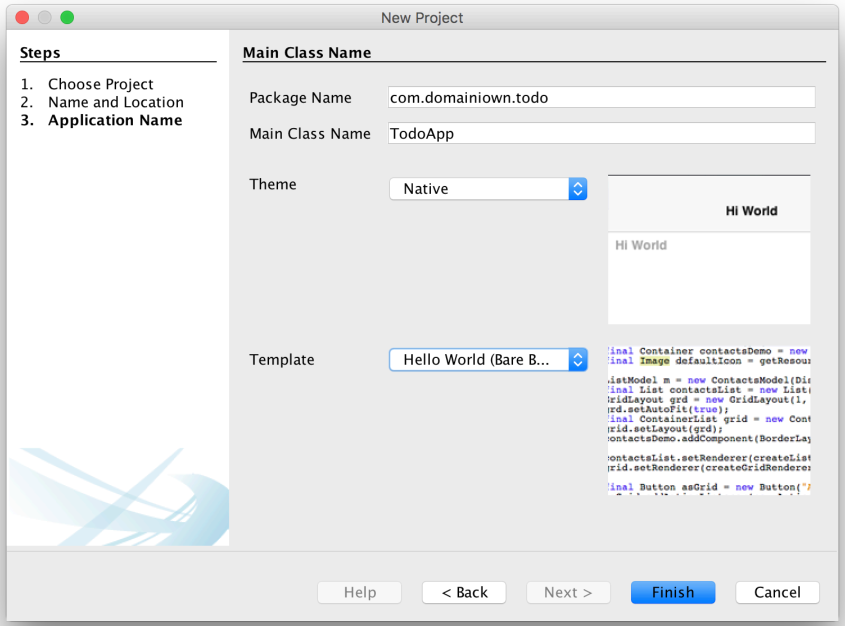


Figure 2. 1. Final Result of the Todo App

I chose to create a similar UI in this case but still respected some platform conventions e.g. notice the title is aligned differently on Android and iOS. This is intentional. I'll discuss this more in the theming section.

I'll start by creating a new project just like the hello world project before but I'll name it "TodoApp".



Package Name: com.domainiown.todo

Main Class Name: TodoApp

Theme: Native

Template: Hello World (Bare B...

```
inal Container contactsDemo = new
inal Image defaultIcon = getResou

istModel m = new ContactsModel(Di
inal List contactsList = new List
ridLayout grd = new GridLayout(1,
rd.setAutoFit(true);
inal ContainerList grid = new Con
rid.setLayout(grd);
ontactsDemo.addComponent(BorderLa

ontactsList.setRenderer(createLis
rid.setRenderer(createGridRendere

inal Button asGrid = new Button("
```

Figure 2. 2. The new Project Wizard for the TodoApp

We'll start by going over the hierarchy of the application.

Layouts and Hierarchy

2.2

Every button, label or element you see on the screen in a Codename One application is a Component. This is a highly simplified version of this class hierarchy:

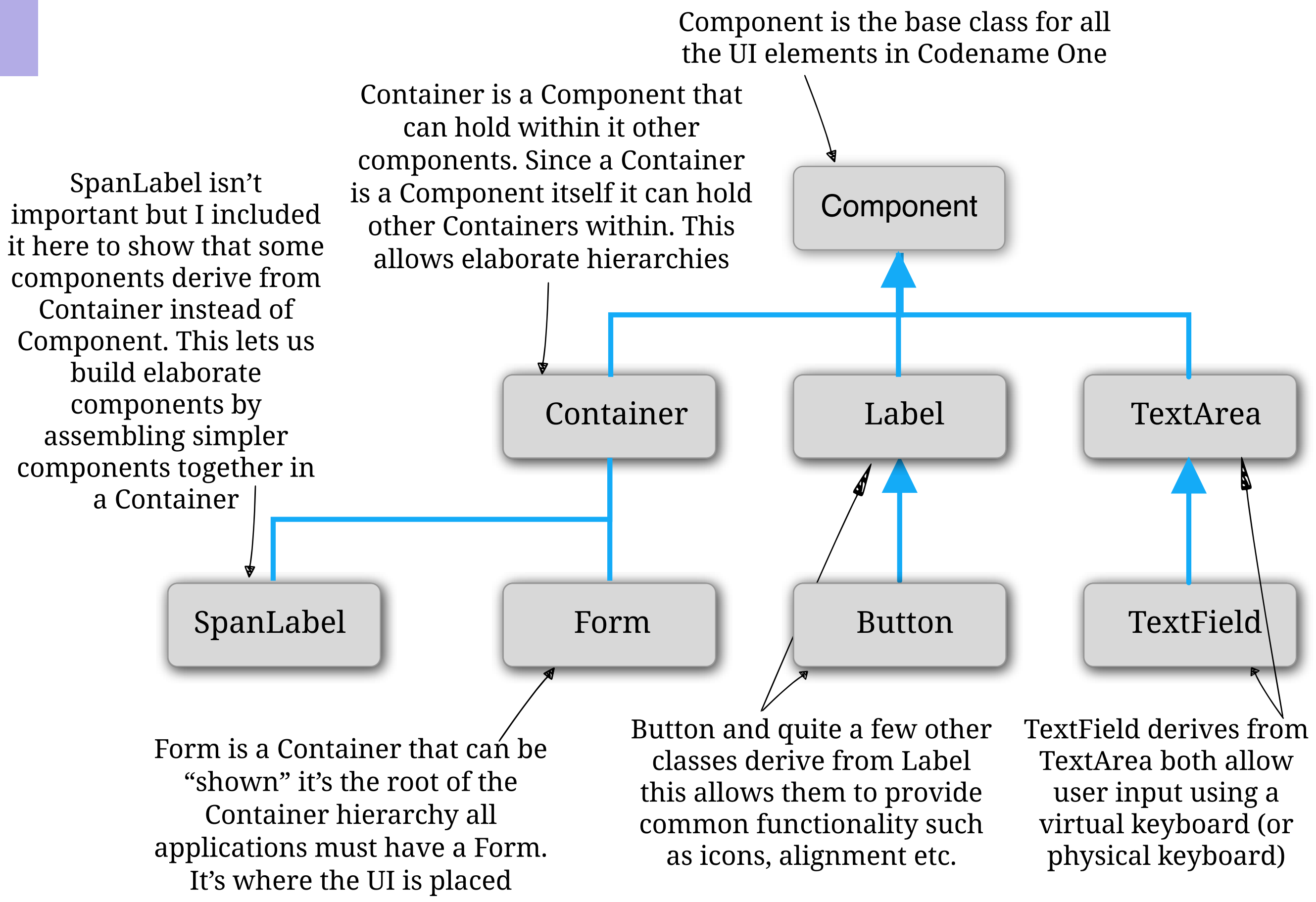

Component is the base class for all the UI elements in Codename One
Container is a Component that can hold within it other components. Since a Container is a Component itself it can hold other Containers within. This allows elaborate hierarchies
SpanLabel isn't important but I included it here to show that some components derive from Container instead of Component. This lets us build elaborate components by assembling simpler components together in a Container
Component
Container
Label
TextArea
SpanLabel
Form
Button
TextField
Form is a Container that can be "shown" it's the root of the Container hierarchy all applications must have a Form. It's where the UI is placed
Button and quite a few other classes derive from Label this allows them to provide common functionality such as icons, alignment etc.
TextField derives from TextArea both allow user input using a virtual keyboard (or physical keyboard)

Figure 2. 3. The Core Component Class Hierarchy

A Codename One application is effectively a series of forms, only one Form can be shown at a time. The Form includes everything we see on the screen. Under the hood the `Form` is comprised of a few separate pieces:

- Content Pane - this is literally the body of the Form. When we add a Component into the Form it goes into the content pane. Notice that Content Pane is scrollable by default on the Y axis!
- Title Area - we can't add directly into this area. The title area is managed by the Toolbar class. Toolbar is a special component that resides in the top portion of the form and abstracts the title design. The title area is broken down into two parts:
  - Title of the Form and its commands (the buttons on the right/left of the title)
  - Status Bar - on iOS the area on the top includes a special space so the notch, battery, clock etc. can fit. Without this the battery indicator/clock or notch would be on top of the title

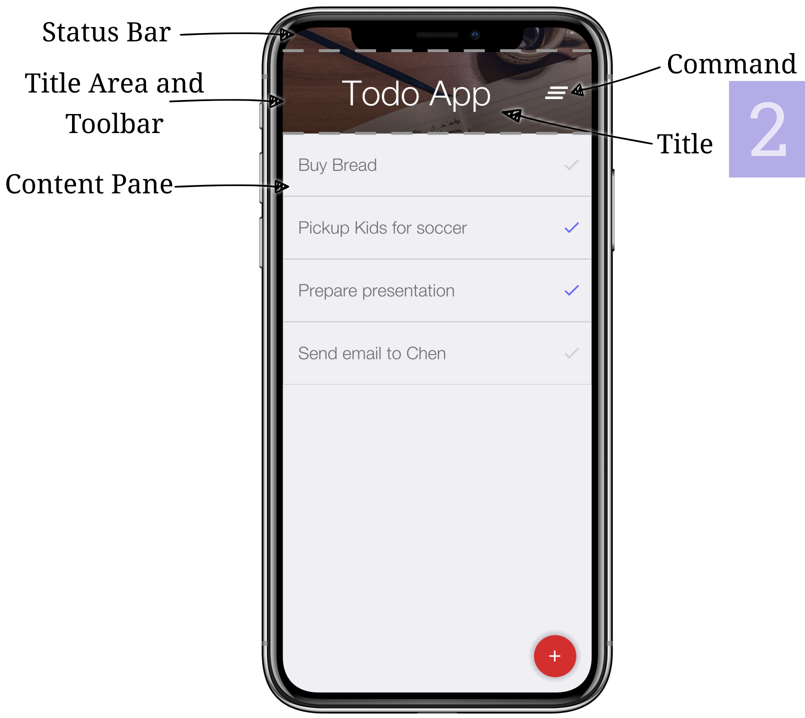


Figure 2. 4. Structure of a Form

Now that we understand this let's look at the new project we created and open the Java file TodoApp.java. In it we should see the lines that setup the UI in the start() method:

Listing 2. 1. The Default Form of TodoApp

```
Form hi  = new Form("Hi World", BoxLayout.y());
hi.add(new Label("Hi World"));
hi.show();
```

This is the same code we had in the hello world app...

I'll circle back to the layout code soon. Right now I want to create a new Form for the todo app.

We can start with something like this:

Listing 2. 2. The Default Form of TodoApp

```
Form todoApp = new Form("Todo App",   BoxLayout.y());

todoApp.show();
```

I changed the title/name and removed the hi world label

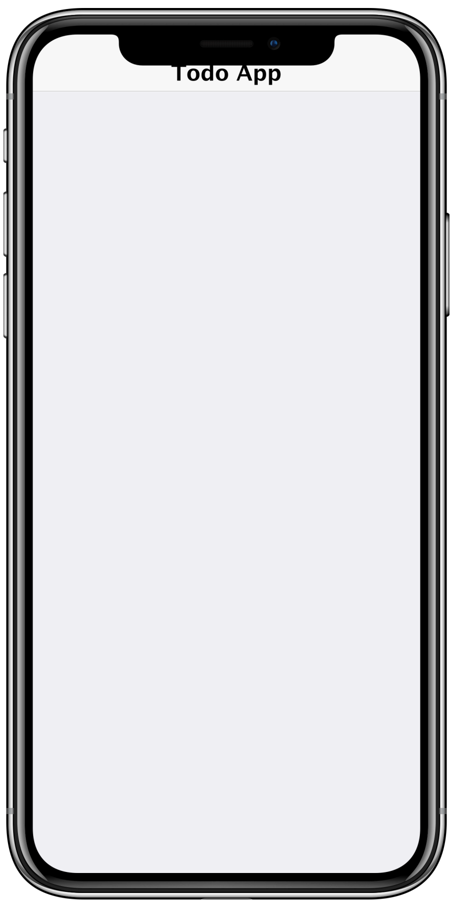


Figure 2. 5. The Initial Step - Not much to see here...

I think it would make more sense to separate the code to a different class. For convenience I will derive the `Form` class. This is a common practice in UI frameworks although it isn't a requirement, e.g. in the hello world app we just used the `Form` without inheriting it. First I'll need to allocate and show the new class in the `TodoApp` class:

Listing 2. 3. TodoApp: Create and Show the new Form

```
new TodoForm().show();
```

Notice I don't need to save the instance of the `TodoForm`, I just show it immediately. I can create a `Form` field but there is no need for that since the `Form` is shown immediately.

Then I can implement the barebones TodoForm class. This obviously needs to reside in a new TodoForm.java file as required for public Java classes:

Listing 2. 4. TodoForm Create and Show the new Form

```
public  class TodoForm extends Form {
    public  TodoForm() {
        super( "Todo App",  BoxLayout. y());
    }
}
```

This code is equivalent to the one we had before specifically new Form("Todo App", BoxLayout.y());

I separated the code to a separate class for convenience so I can encapsulate specific functionality. Specifically API's such as adding items, the floating action button etc.

The second step is adding entries into the todo list. For that I'll use the FloatingActionButton (AKA FAB), this is a staple of Google's material design.

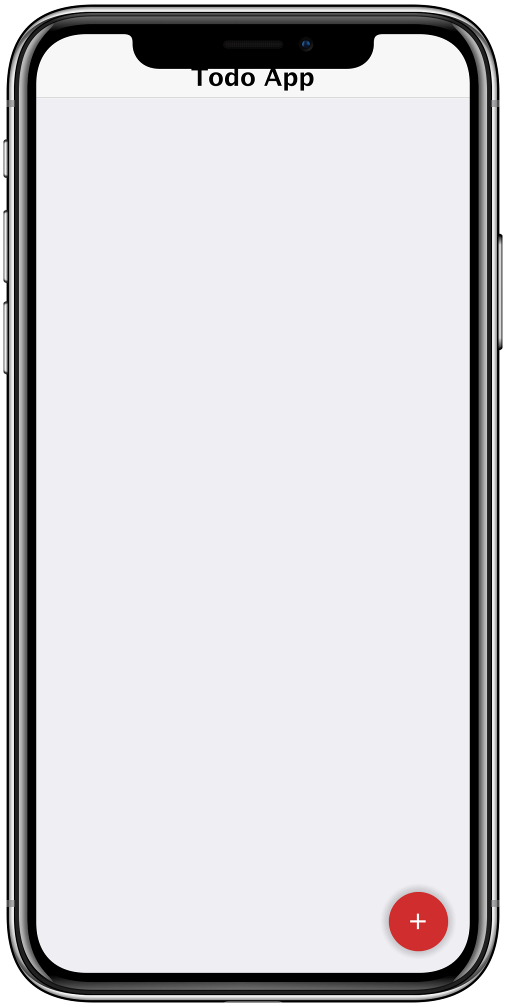


Figure 2. 6. Step 2 - The Floating Action Button That Adds Entries

Material design is Google's all encompassing UI design paradigm. It specifies how all UI elements should look/behave. It's the UI standard on Android. The Uber app uses its principles on iOS and on Android

Before I add this I'll add a stub method so the FloatingActionButton can trigger an event:

Listing 2. 5. TodoForm.addNewItem() Stub

```
private  void  addNewItem()  {}
```

We'll fill this method soon...

Now I can add this to the TodoForm constructor using the code:

Listing 2. 6. TodoForm Constructor Floating Action Button

```
FloatingActionButton fab  = FloatingActionButton.
      createFAB(FontImage.MATERIAL_ADD);

fab.bindFabToContainer(this);

fab.addActionListener(e -> addNewItem());
```

FloatingActionButton creates the round floating button, it accepts a FontImage constant which I'll cover soon

We usually add items to containers but a FAB floats on top, this method handles the "floating" aspect. this means the current Form instance.

When the FAB is clicked we invoke the addNewItem() method, I'll discuss event handling soon

FontImage and Material Icons

Material design defines several standard icons that you can find here: material.io/icons/

These icons are integrated into Codename One via an icon font. That effectively means they take up very little RAM and can adapt in terms of size/color without a problem. They are used extensively in the code because they are so convenient and powerful.

The next step would be implementing the addNewItem method that we added before.

Listing 2. 7. TodoForm.addNewItem()

We create a new object instance for each item, we'll discuss that soon

```
private   void  addNewItem() {
     TodoItem td  = new TodoItem("" ,  false);

     add(td);

     revalidate ();

     td.edit();
}
```

We invoke Form's add method to add the item to the UI

When a Form is showing and we make a change to it we need to let the Form know we finished making changes by invoking revalidate()

We launch the device virtual keyboard so the user can start typing the text into the new item immediately

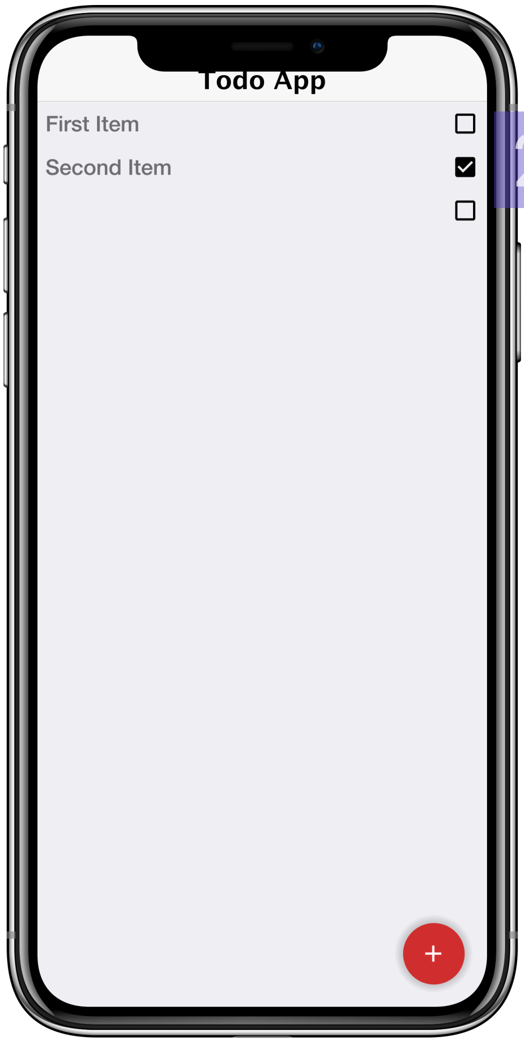


Figure 2. 7. Step 3 - Add a New Item

You won't see the virtual keyboard on the simulator since you would use your desktop's keyboard but, when running on the device, the keyboard would pop up instantly and let you type.

The most confusing aspect in the code would be the revalidate() call. Why do we need it and why doesn't Codename One revalidate every time we add an item?

revalidate() is an expensive operation. If Codename One did it every time the UI changed it would be very slow. In fact platforms such as web do it all the time (reflow) and that's considered one of the major performance penalties inherent in web development. The revalidate operation is expensive since it needs to recursively loop through all components, if one of them changes its size we need to rerun this recursive loop over again. There are shortcuts to make it faster but it's still an inherently slow process.

We can clarify this with an example. Lets say we have this code:

Listing 2. 8. Why Revalidate Matters

```
add(componentX);
// possibly some processing
add(componentY);
```

With automatic revalidate (reflow) we'd have to run revalidate twice which would make things slow.

Another benefit of `revalidate()` is layout animations. We can change the UI and instead of revalidate we can ask Codename One to animate the UI into place e.g.:

Listing 2. 9. Layout Animation

```
add(componentX);
animateLayout(150);
```

This will move `componentX` into it's place in the layout within 150 milliseconds. It's equivalent to revalidate in effect but does that logic with "style". I'll discuss layout animations again later in this chapter.

Before that lets look at the `TodoItem` class:

Listing 2. 10. The TodoItem class

```
public  class  TodoItem extends Container {
    private  TextField  nameText;
    private  CheckBox done = new CheckBox();
    public  TodoItem(String name, boolean checked) {
        super( new BorderLayout ());
        nameText= new TextField (name);
        nameText.setUIID ( "Label" );
        add(CENTER, nameText);
        add(EAST,  done);
        done.setSelected(checked);
    }
    public  void  edit ()  {
        nameText.startEditingAsync();
    }
    public  boolean  isChecked()  {
        return  done.isSelected ();
    }
    public  String  getText()  {
        return  nameText.getText();
    }
}
```

Inheriting Container makes it easy to detect if a TodoItem is checked

TextField accepts user text input with the virtual keyboard

CheckBox can be toggled between selected and unselected states

With BorderLayout we can position a component in one of 5 places: CENTER, EAST, WEST, NORTH and SOUTH

We don't want the text field to look like a text field, we want it to look like a label which we can do by setting the UIID We'll discuss this in the styling section soon

We place the text field in the center of the layout area and the check box to the right (EAST)

We'll need the last two method when we'll save the data to to device storage (flash)

This launches the text editing, on the device and opens the virtual keyboard

The BorderLayout constants and other constants are defined in the com.codename1.ui.CN class which we import statically into every Java source file with the code import static com.codename1.ui.CN.*; I assume in the code that this static import exists in all client side files!

This brings me to the discussion of layout managers. We've used two so far: BoxLayout and BorderLayout Let's dig deeper.

2.2.1 Layout Managers

A layout manager is an algorithm that decides the size and location of the components within a Container . Every Container has a layout manager associated with it. The default layout manager is FlowLayout.

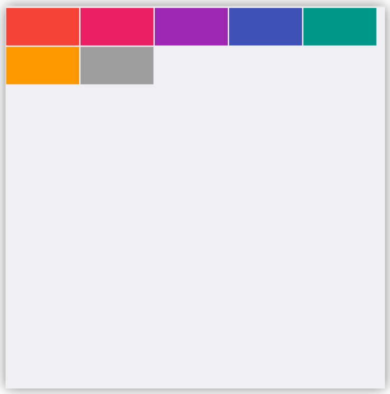
To understand layouts we need to understand a basic concept about Component. Each component has a “preferred size”. This is the size in which a component “wants” to appear. E.g. for a Label the preferred size will be the exact size that fits the label text, icon and padding of the component.

A layout manager places a component based on its own logic and the preferred size (sometimes referred to as “natural size”). A FlowLayout will just traverse the components based on the order they were added and size/place them one after the other. When it reaches the end of the row it will go to the new row.

Use* FlowLayout *Only for Simple Things

FlowLayout is great for simple things but has issues when components change their sizes dynamically (like a text field). In those cases it can make bad decisions about line breaks and take up too much space

Flow Layout sizes components based on their preferred size and arranges them from left to right. When we reach the end of the line it breaks a line



Flow layout has several modes including a center mode that center aligns elements. It can align elements to the right and align vertically as well

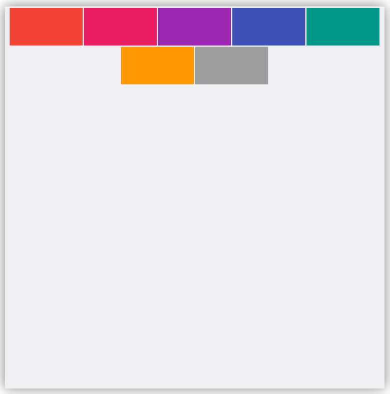
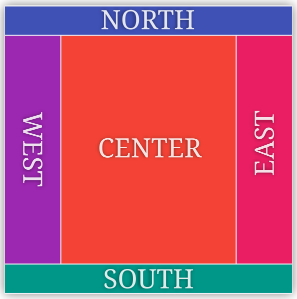
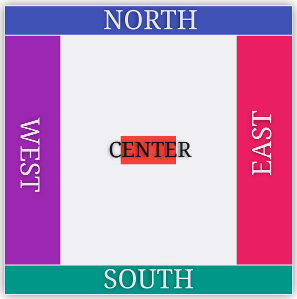


Figure 2. 8. Layout Manager Primer Part I

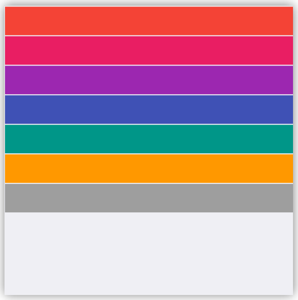
Border Layout can position elements in the NORTH, SOUTH, EAST, WEST and CENTER. Components take their preferred size on the opposing axis. Center takes up the available space by default



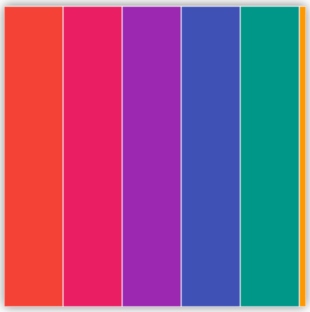
Border Layout has several modes including absolute center mode where the center component takes up only its preferred size



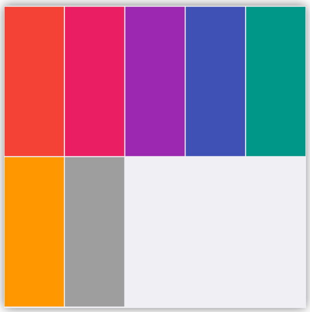
BoxLayout Y arranges components vertically, giving them the available width and their preferred height



BoxLayout X arranges components horizontally, giving them the available height and their preferred width



GridLayout arranges components in a grid and gives every element the same size to match the preferred size of the largest elements



LayeredLayout places components one on top of the other. They have some spacing here so you can see the layers below

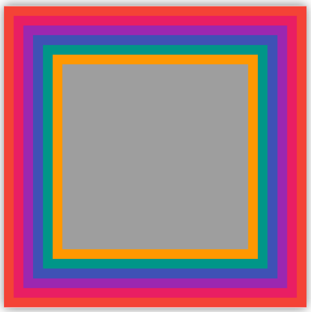


Figure 2. 9. Layout Manager Primer Part II

There are a few other interesting layouts in Codename One such as `TableLayout`, `GridBagLayout`, `MigLayout` etc. but I don't use them in the book

Scrolling doesn't work well for all types of layouts as the positioning algorithm within the layout might break. Scrolling on the Y axis works great for `BoxLayout` Y which is why I picked it for the `TodoForm`:

Table 2. 1. Scrolling in Layout Managers

Layout	Scrollable
Flow Layout	Possible on Y axis only
Border Layout	Scrolling is blocked
Box Layout Y	Scrollable only on the Y axis

Layout	Scrollable
Box Layout X	Scrollable only on the X axis
Grid Layout	Scrollable
LayeredLayout	Not scrollable (usually)

Nesting Scrollable Containers

Only one element can be scrollable within the hierarchy, otherwise if you drag your finger over the Form Codename One won't know which element you are trying to scroll. By default form's content pane is scrollable on the Y axis unless you explicitly disable it (setting the layout to BorderLayout implicitly disables scrolling).

It's important to notice that it's OK to have non-scrollable layouts, e.g. BorderLayout, as items within a scrollable container type. E.g. in the TodoApp we added TodoItem which uses BorderLayout into a scrollable BoxLayout Form.

Layouts can be divided into two distinct groups:

- Constraint Based - BorderLayout (and a few others I won't discuss in this book such as GridBagLayout, MigLayout and TableLayout)
- Regular - All of the other layout managers

When we add a Component to a Container with a regular layout we do so with a simple add method:

Listing 2. 11. Adding to a Regular Container

```
cnt.add(new Label("Just Added"));
```

This works great for regular layouts but might not for constraint based layouts. A constraint based layout accepts another argument. E.g. BorderLayout needs a location for the Component:

Listing 2. 12. Adding to a Regular Container

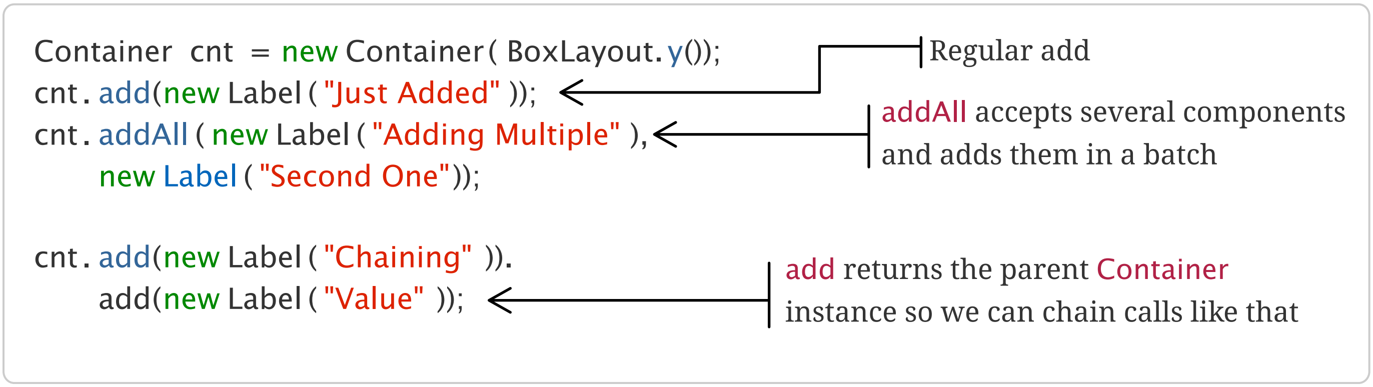
```
cnt.add(NORTH,new Label("Just Added"));
```

This line assumes you have an import static com.codename1.ui.CN.*; in the top of the file. In BorderLayout (which is a constraint based layout) placing an item in the NORTH places it in the top of the Container.

Terse Syntax

Almost every layout allows us to add a component using several variants of the add method:

Listing 2. 13. Versions of add



```
Container cnt = new Container( BoxLayout.y());
cnt.add(new Label ("Just Added" ));
cnt.addAll ( new Label ( "Adding Multiple" ),
        new Label ("Second One"));

cnt.add(new Label ("Chaining" )).
        add(new Label ("Value" ));
```

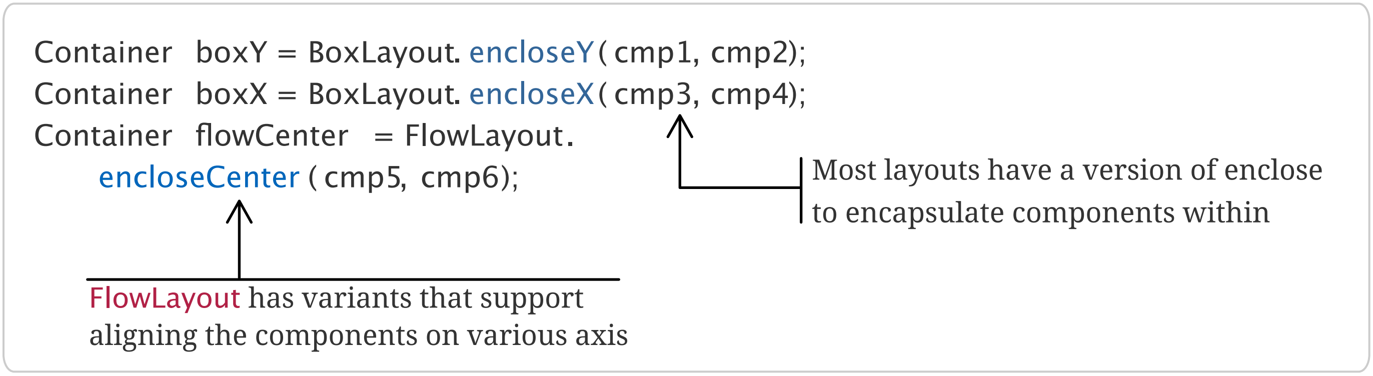
Regular add

addAll accepts several components and adds them in a batch

add returns the parent Container instance so we can chain calls like that

In the race to make code "tighter" we can make this even shorter. Almost all layout managers have their own custom terse syntax style e.g.:

Listing 2. 14. Terse Syntax



```
Container boxY = BoxLayout. encloseY( cmp1, cmp2);
Container boxX = BoxLayout. encloseX( cmp3, cmp4);
Container flowCenter = FlowLayout.
        encloseCenter ( cmp5, cmp6);
```

Most layouts have a version of enclose to encapsulate components within

FlowLayout has variants that support aligning the components on various axis

To sum this up, we can use layout managers and nesting to create elaborate UI's that implicitly adapt to different screen sizes and device orientation.

Styles 2.2.2

The next stage in the evolution of the application is making it look good. To understand what that means I need to introduce you to 3 important terms in Codename One: Theme, Style and UIID.

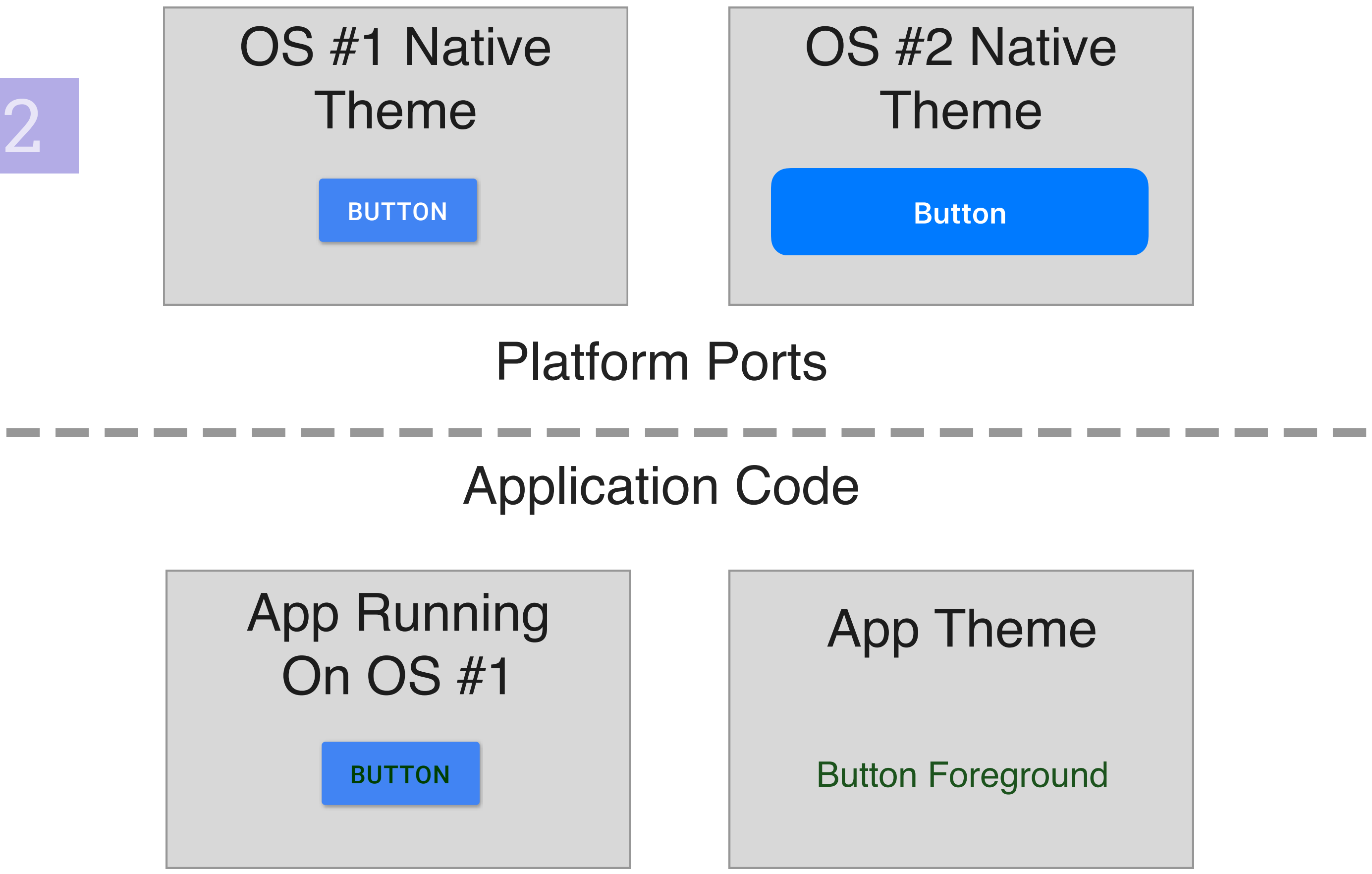


Figure 2. 10. Themes as Layers

Themes are very similar conceptually to CSS, in fact they can be created with CSS syntax as explained in Appendix C (page 399). The various Codename One ports ship with a native theme representing the appearance of the native OS UI elements. Every Codename One application has its own theme that derives the native theme and overrides behavior within it.

If the native theme has a button defined, we can override properties of that button in our theme. This allows us to customize the look while retaining some native appearances. This works by merging the theme to one big theme where our application theme overrides the definitions of the native theme. This is pretty similar to the cascading aspect of CSS if you are familiar with that.

Themes consist of a set of UIID definitions. Every component in Codename One has a UIID associated with it. UIID stands for User Interface Identifier. This UIID connects the theme to a specific component. You may recall we wrote this code before:

Listing 2. 15. setUIID on TextField

```
nameText.setUIID("Label");
```

$\longleftarrow$ This is a text field component that will look like a Label

Effectively we told the text field that it should use the UIID of Label when it's drawing itself. That way the text field looks like a Label. It's very common to do tricks like that in Codename One. E.g. button.setUIID("Label") which would make a button appear like a label and allow us to track clicks on a "Label".

The UIID's translate the theme elements into a set of Style objects. These Style objects get their initial values from the theme but can be further manipulated after the fact. So if I want to make the text field's foreground color red I could use this code:

Listing 2. 16. setUIID on TextField

```
nameText.getAllStyles().setFgColor(0xff0000);
```

The color is in hexadecimal RRGGBB format so 0xff00 would be green and 0xff0000 would be red.

getAllStyles() returns a Style object but why do we need "all" styles?

Each component can have one of 4 states and each state has a Style object. This means we can have 4 style objects per Component:

- **Unselected** - used when a component isn't touched and doesn't have focus. You can get that object with getUnselectedStyle() .
- **Selected** - used when a component is touched or if focus is drawn for non-touch devices. You can get that object with getSelectedStyle() .
- **Pressed** - used when a component is pressed. Notice it's only applicable to buttons and button subclasses usually. You can get that object with getPressedStyle().
- **Disabled** - used when a component is disabled. You can get that object with getDisabledStyle().

The getAllStyles() method returns a special case Style object that lets you set the values of all 4 styles from one class so the code before would be equivalent to invoking all 4 setFgColor methods. However, getAllStyles() only works for setting properties not for getting them!

⚠ Don't use getStyle() for manipulation. getStyle() returns the current Style object which means it will behave inconsistently. The paint method uses getStyle() as it draws the current state of the Component but other code should avoid that method. Use the specific methods instead: getUnselectedStyle(), getSelectedStyle(), getPressedStyle() , getDisabledStyle() and getAllStyles()

As you can see, it's a bit of a hassle to change styles from code which is why the theme is so appealing.

Designer Tool

As I mentioned before, we can customize the theme using a CSS like syntax which I discuss in Appendix C (page 399). Here I'll explain the usage of Codename One Designer and the simulator theming tools to customize the look of components.

The theme is stored in the `theme.res` file in the `src` root of the project. We load the theme file using this line of code in the `init(Object)` method in the main class of the application:

Listing 2. 17. Theme Loading Code

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

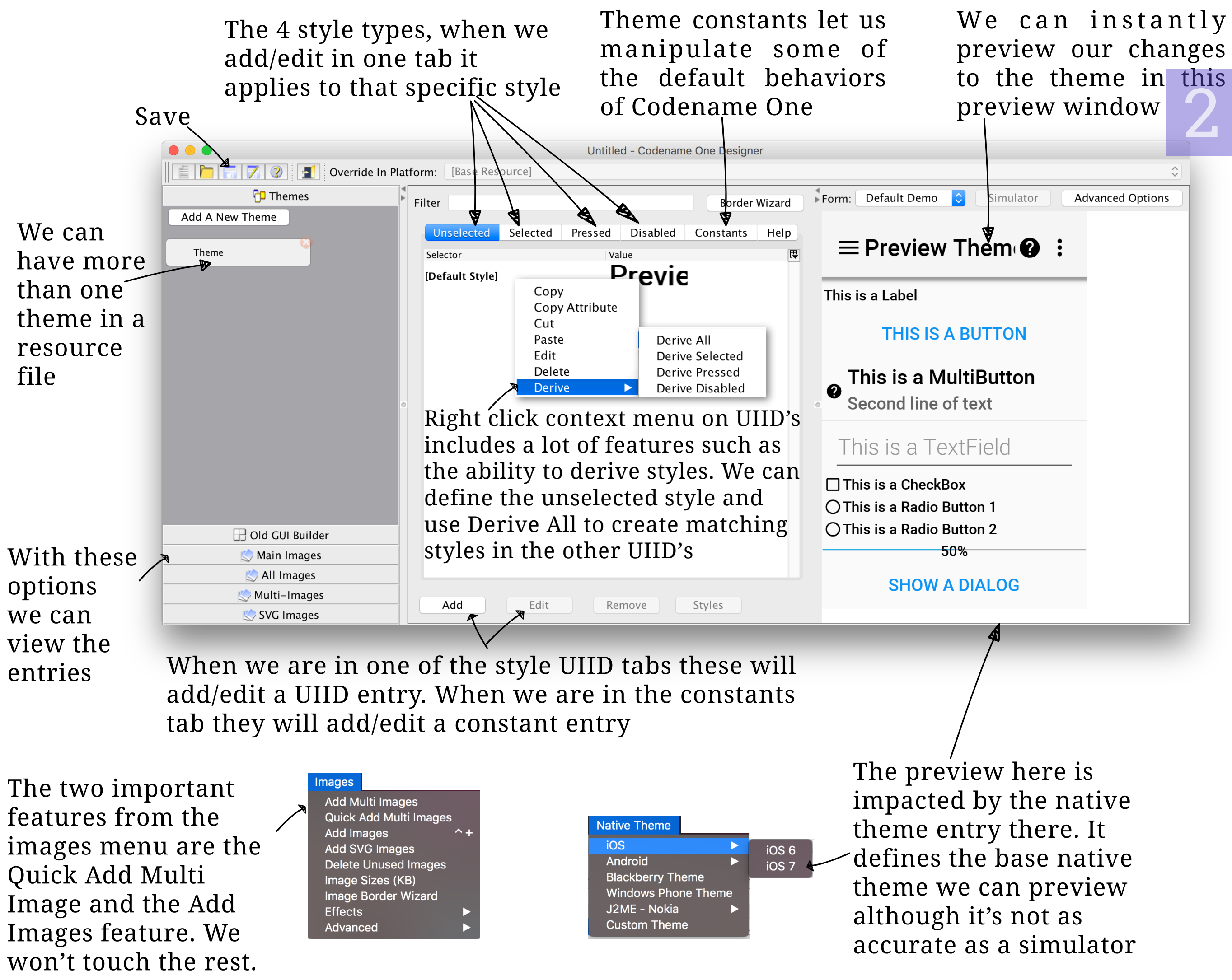
This code is shorthand for resource file loading and for the installation of theme. You could technically have more than one theme in a resource file at which point you could use `initNamedTheme()` instead. The resource file is a special file format that includes inside it several features:

- Themes
- Images
- Localization Bundles
- Data files

It also includes some legacy features such as the old GUI builder.

> I'm mentioning these for reference only I don't discuss the new/old GUI builders in this book

We can open the designer tool by double clicking the res file. The UI can be a bit overwhelming at first so I'll try to walk slowly through the steps.



Save

The 4 style types, when we add/edit in one tab it applies to that specific style

Theme constants let us manipulate some of the default behaviors of Codename One

We can instantly preview our changes to the theme in this preview window

We can have more than one theme in a resource file

Right click context menu on UIID's includes a lot of features such as the ability to derive styles. We can define the unselected style and use Derive All to create matching styles in the other UIID's

With these options we can view the entries

When we are in one of the style UIID tabs these will add/edit a UIID entry. When we are in the constants tab they will add/edit a constant entry

The two important features from the images menu are the Quick Add Multi Image and the Add Images feature. We won't touch the rest.

The preview here is impacted by the native theme entry there. It defines the base native theme we can preview although it's not as accurate as a simulator

Figure 2. 11. Codename One Designer Feature Map

Lets start with something simple, like the title design of the app. For this we will need the todo-title.jpg file or equivalent. In my case the file is an 800x356 image but any reasonably sized image with the right colors will do.

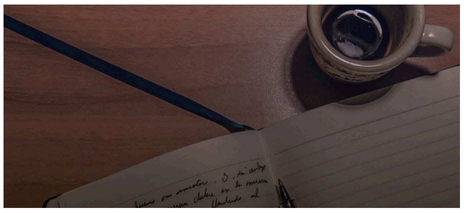


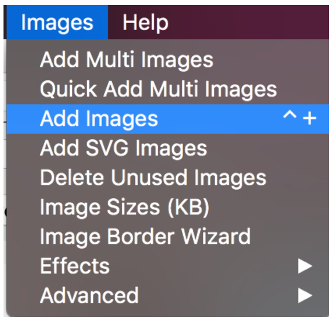
Figure 2. 12. todo-title.jpg

The first step is to add the image. In the designer tool we go to the Images → Add Image menu. Pick your image in the file chooser.

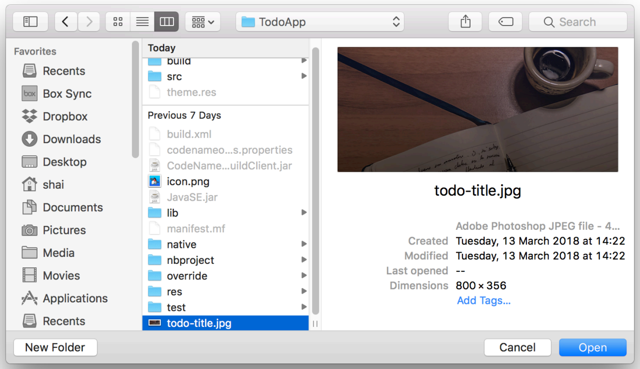
2

Make sure to pick a JPEG or PNG image. It shouldn't be **huge**, 1024px wide would be plenty for this. Notice that if you pick a different format (e.g. Jpeg2000, tiff, svg, gif etc.) it might not work. Some images, specifically those from cell phone cameras, include orientation information and that would be ignored. If your image is flipped, make sure to edit it first

1 Select the Images menu and click Add Images



2 Pick the image in the file picker dialog box



3 You should be able to see the image in the Main Images section

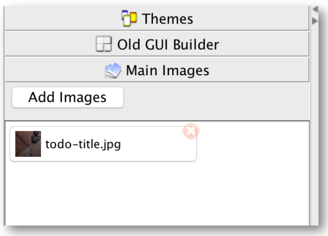


Figure 2. 13. Add a New Image

Multi Images

As I mentioned before, images look radically different on devices with different densities. A common solution is to bundle multiple versions of the image; One for each DPI. Codename One takes that same approach and also simplifies it with multi-images.

A Multi Image is a concept that exists only within the resource file. When we read a Multi Image from the file, it's indistinguishable from a regular image. During the resource file loading process only the image closest to the current DPI is loaded and the rest of the images are skipped.

This begs the question: why not use a regular image and just scale it?

- Scaling images on the device produces some artifacts in scaling as the high quality scaling algorithms are very slow
- To scale we'd want the largest possible image as scaling down is superior to scaling up, this would be memory intensive

Still there is an obvious tradeoff of application size when working with Multi Images so use this feature with care.

Codename One supports the following densities for multi-images:

Table 2. 2. Densities

Constant	Density	Example Device
`DENSITY_VERY_LOW`	~ 88 ppi	
`DENSITY_LOW`	~ 120 ppi	Android ldpi devices
`DENSITY_MEDIUM`	~ 160 ppi	iPhone 3GS, iPad, Android mdpi devices
`DENSITY_HIGH`	~ 240 ppi	Android hdpi devices
`DENSITY_VERY_HIGH`	~ 320 ppi	iPhone 4, iPad Air 2, Android xhdpi devices
`DENSITY_HD`	~ 540 ppi	iPhone 6+, Android xxhdpi devices
`DENSITY_560`	~ 750 ppi	Android xxxhdpi devices
`DENSITY_2HD`	~ 1000 ppi	
`DENSITY_4K`	~ 1250ppi	

The most common devices at the time of this writing (mid 2018) range between DENSITY_HIGH and DENSITY_560

To add a Multi Image we usually create the resource for a high DPI device we then use the menu option Images → Quick Add Multi Image to pick said image and select its DPI from the list of DPI's. E.g. we can ask our designer for a resource designed for the Pixel 2 XL which is a DENSITY_HD device. Then use the Quick Add Multi Image menu and select DENSITY_HD in the following prompt.

The system automatically generates all the other DPI's by scaling down from the DENSITY_HD image and scaling up to the higher DPI's. You can then select the Multi Image entry in the designer and customize individual images within if necessary.

Since scaling is done on the desktop it uses the high quality scale algorithm. Notice you can manually customize individual resolutions of the Multi Image as well.

Now we can go back to the theme view in the designer tool and press the Add button in the Unselected tab.

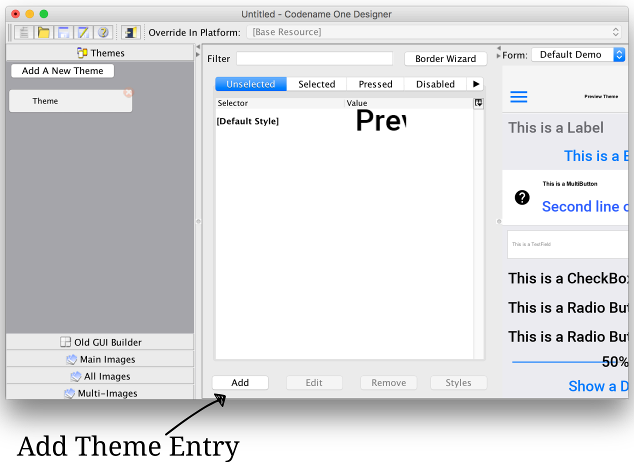
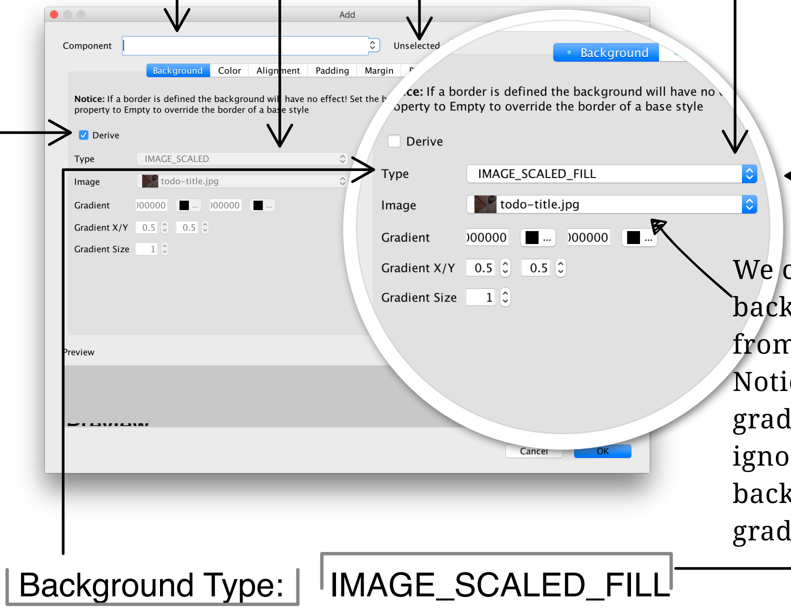


Figure 2. 14. The Add Button

After pressing that button we should see something that looks like this:

To set the background image of the Toolbar we need to uncheck "Derive"

We are adding an entry in the Unselected



Component

Unselected

Notice: If a border is defined the background will have no effect! Set the border property to Empty to override the border of a base style

Derive

Type IMAGE_SCALED_FILL

Image todo-title.jpg

Gradient 000000 000000

Gradient X/Y 0.5 0.5

Gradient Size 1

Background Type: IMAGE_SCALED_FILL

We can pick the background image from the combo box. Notice that the gradient values are ignored unless the background type is a gradient

There are many background type options. Here I pick IMAGE_SCALED_FILL which will scale the image to take up all the available space and keep aspect ratio

The UIID we are adding or editing. You can type in any arbitrary name when adding. You can also pick from pre-existing options in the combo box. Right now we can type Toolbar here

When Derive is checked we derive this specific attribute from the native theme

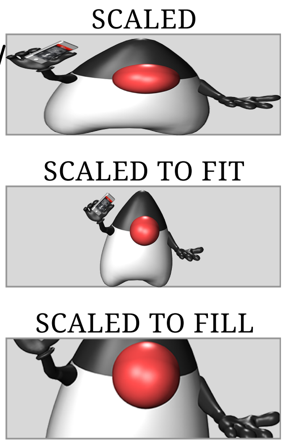
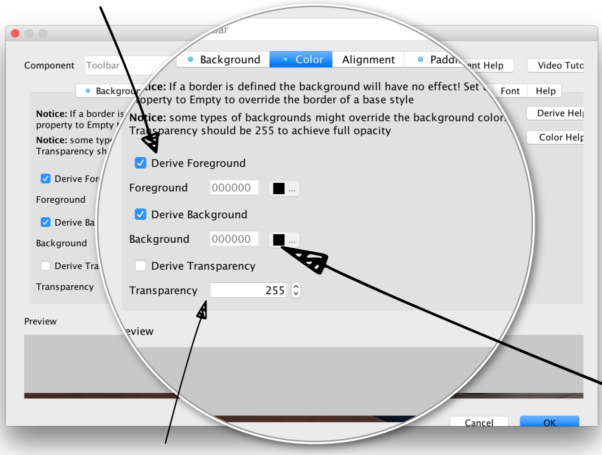


Figure 2. 15. Add the Theme Entry for the Toolbar

Don't forget to press the save button in the designer after making changes

There are several other options in the add theme entry dialog. Lets go over them and review what we should do for each tab in this UI:

The foreground color of the component e.g. the text color of a label in this case we don't need the foreground as we'll style it in the "Title" UIID



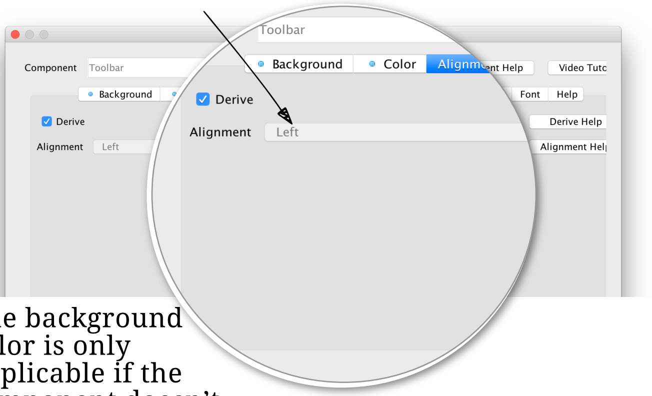
Notice: If a border is defined the background will have no effect! Set property to Empty to override the border of a base style

Notice: some types of backgrounds might override the background color. Transparency should be 255 to achieve full opacity

- [✓] Derive Foreground
- Foreground: 000000
- [✓] Derive Background
- Background: 000000
- [ ] Derive Transparency
- Transparency: 255

Transparency of 255 indicates completely opaque and 0 indicates complete transparency. It's best to define it to 255 as the image we show is opaque

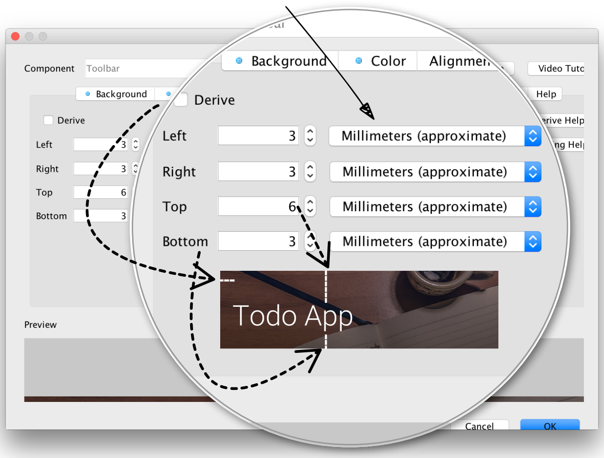
Alignment can be left/right or center. This isn't applicable to all components and will only work for components deriving from Label or TextArea.



- [✓] Derive
- Alignment: Left

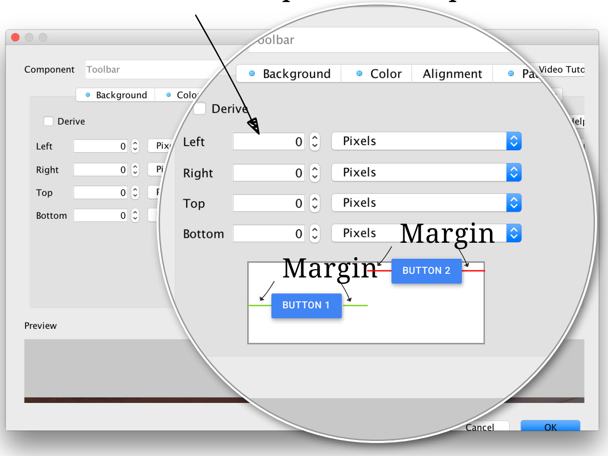
The background color is only applicable if the component doesn't have a border and doesn't have a background type

Padding is the extra space the component takes beyond its "natural size". It can be expressed in millimeters, pixels or percentage of the screen size. We almost always use millimeters for padding. Notice that in the screenshot below I ignore the right margin as the title on Android is aligned to the left



- [ ] Derive
- Left: 3 Millimeters (approximate)
- Right: 3 Millimeters (approximate)
- Top: 6 Millimeters (approximate)
- Bottom: 3 Millimeters (approximate)

Margin is the space between this component and the other components next to it. We often set it to 0 when we want to take up available space

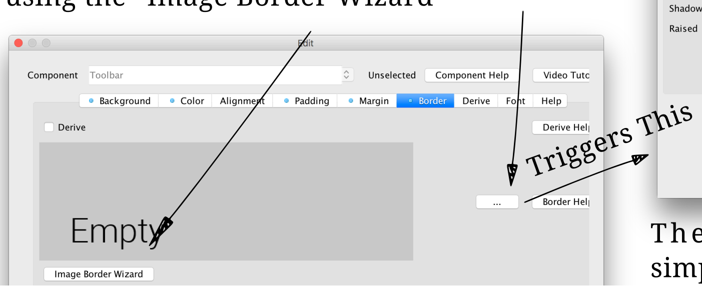


- [ ] Derive
- Left: 0 Pixels
- Right: 0 Pixels
- Top: 0 Pixels
- Bottom: 0 Pixels

Figure 2. 16. The Rest of the Add Theme Entry Dialog - Part I

When a border is defined it overrides the background and color. That's why it's important to define it as "Empty" sometimes. The Toolbar has a shadow border defined on Android. We want to disable that so our background image will show. We can edit the border with the "..." button and we can create a 9-piece border using the "Image Border Wizard"

This is a round border we use in the FloatingActionButton



Edit

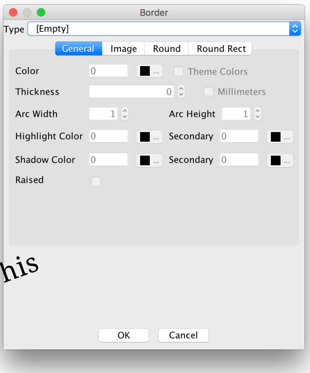
Component: Toolbar — Unselected — Component Help — Video Tutc

Background · Color · Alignment · Padding · Margin · Border · Derive · Font · Help

Derive — Derive Help

Empty — ... — Border Help

Image Border Wizard



Border

Type: [Empty]

General · Image · Round · Round Rect

Color: 0 — Theme Colors

Thickness: 0 — Millimeters

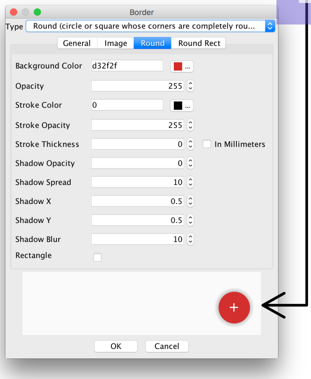
Arc Width: 1 — Arc Height: 1

Highlight Color: 0 — Secondary: 0

Shadow Color: 0 — Secondary: 0

Raised

OK — Cancel



Border

Type: Round (circle or square whose corners are completely rou...

General · Image · Round · Round Rect

Background Color: d32f2f

Opacity: 255

Stroke Color: 0

Stroke Opacity: 255

Stroke Thickness: 0 — In Millimeters

Shadow Opacity: 0

Shadow Spread: 10

Shadow X: 0.5

Shadow Y: 0.5

Shadow Blur: 10

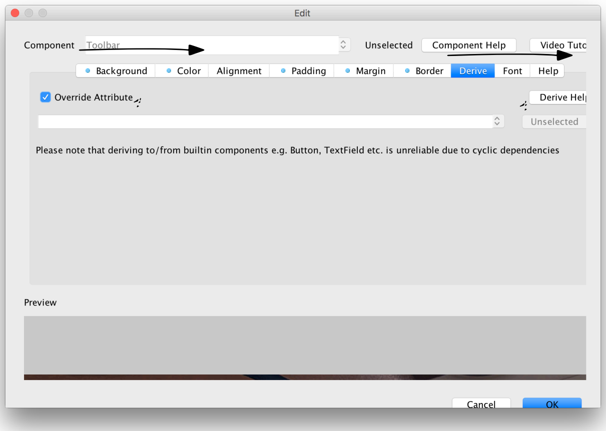
Rectangle

OK — Cancel

There are multiple simple border types in the border editing dialog but the ones we use most of all are the Round, RoundRect, Line, Underline and Image (AKA 9-piece)

When we select the Round Border we can create either a circle or a pill shape (by activating the rectangle flag)

Derive inherits the styling of the given UIID you can type the UIID on the left and select the right state on the right



Edit

Component: Toolbar — Unselected — Component Help — Video Tutc

Background · Color · Alignment · Padding · Margin · Border · Derive · Font · Help

Override Attribute [✓] — Derive Help

Unselected

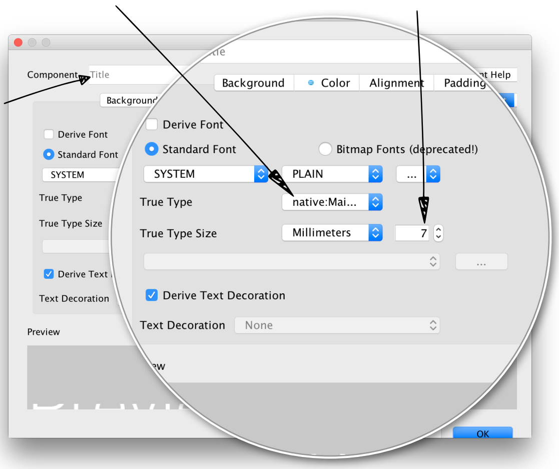
Please note that deriving to/from builtin components e.g. Button, TextField etc. is unreliable due to cyclic dependencies

Preview

Cancel — OK

The Font UI is somewhat confusing due to a heavy dose of legacy features. For modern applications it makes sense to pick a native true type font from the combo box here and size it in millimeters

Toolbar doesn't need a font since it doesn't have text so I chose to show the font styling for Title because fonts are important



Component: Title

Background · Color · Alignment · Padding · Help

Derive Font

Standard Font [✓] — Bitmap Fonts (deprecated!)

SYSTEM — PLAIN — ...

True Type: native:Mai...

True Type Size: Millimeters — 7

Derive Text Decoration [✓]

Text Decoration: None

Preview

OK

Figure 2. 17. The Rest of the Add Theme Entry Dialog - Part II

Let's pause for a moment. We can't go through the entire book and post images like this for every UIID we run into… It would turn into a picture book!

Furthermore, you might prefer to work with CSS instead of using the designer tool. So we need a more concise way to express the settings in the designer tool. In this case I can write the settings for the `Toolbar` UIID like this:

Listing 2. 18. Toolbar Styling

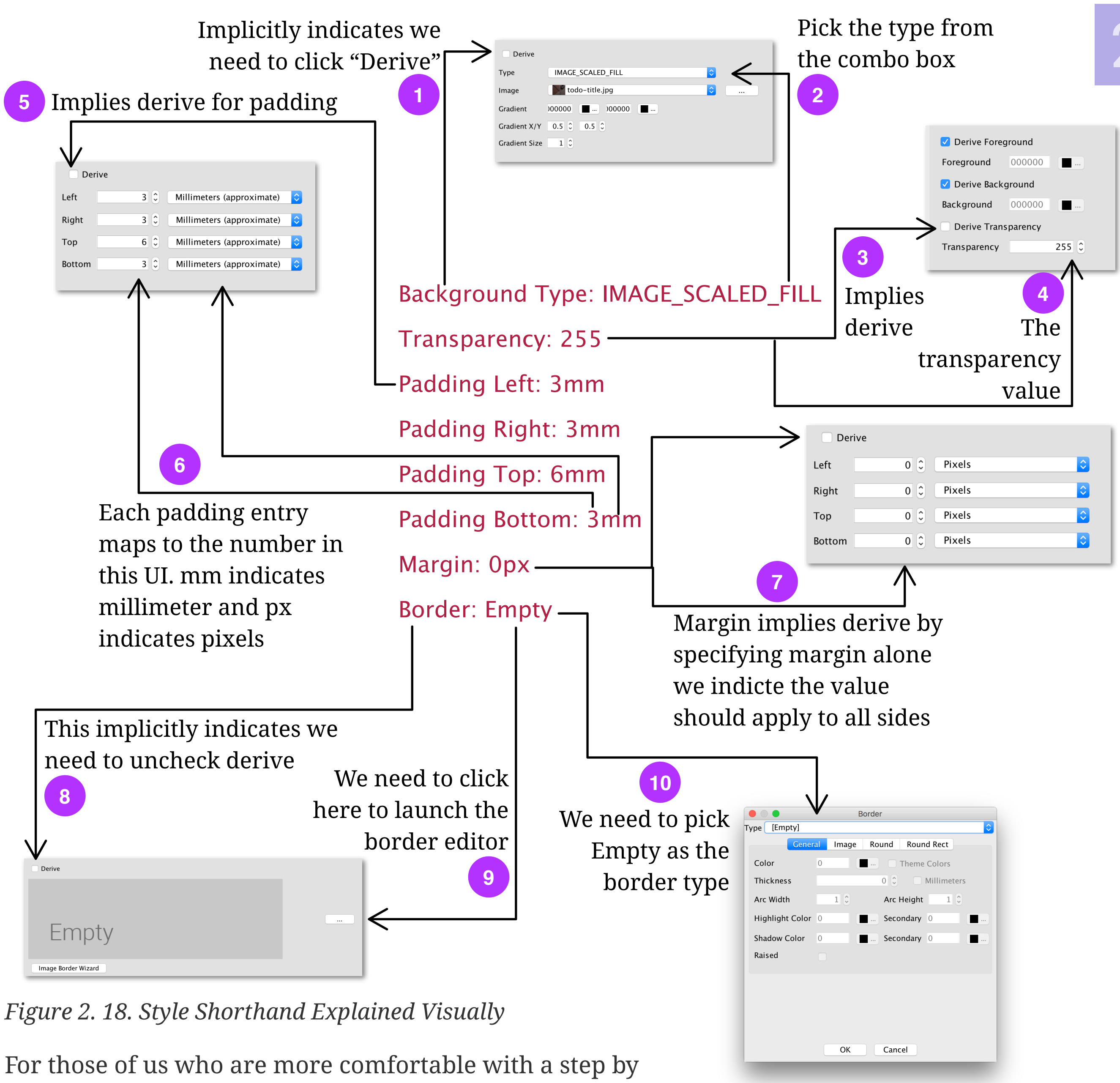
```
Background Type: IMAGE_SCALED_FILL
Transparency: 255
Padding Left: 3mm
Padding Right: 3mm
Padding Top: 6mm
Padding Bottom: 3mm
Margin: 0px
Border: Empty
```

Notice I enumerate the padding for each side

Since the margin is identical on all sides I only list it as margin

I ignore everything that I didn't derive. This is far more concise and in some regards much simpler than the images!

If you are still in doubt, you can use the project sources and open the file using the designer tool to review the settings in the resource file.



Implicitly indicates we need to click "Derive"
1

Pick the type from the combo box
2

5 Implies derive for padding

3 Implies derive

4 The transparency value

Background Type: IMAGE_SCALED_FILL

Transparency: 255

Padding Left: 3mm

Padding Right: 3mm

Padding Top: 6mm

Padding Bottom: 3mm

Margin: 0px

Border: Empty

6 Each padding entry maps to the number in this UI. mm indicates millimeter and px indicates pixels

7 Margin implies derive by specifying margin alone we indicte the value should apply to all sides

This implicitly indicates we need to uncheck derive
8

We need to click here to launch the border editor
9

10 We need to pick Empty as the border type

Figure 2. 18. Style Shorthand Explained Visually

For those of us who are more comfortable with a step by step guide this is the exact process to produce this style:

- Double click the theme.res file in the src directory to open the designer tool
- Click Theme on the left hand side
- Click the Add button on the bottom of the screen. You should see the Add dialog box with the Background tab selected

```
Background Type: IMAGE_SCALED_FILL
```

- Uncheck the Derive check box
- Pick IMAGE_SCALED_FILL in the Type combo box. This assumes you already added a single image to the theme, it should be selected already in the Image combo box

```
Transparency: 255
```

- Switch to the Color tab
- Uncheck the Derive Transparency checkbox
- Type into the Transparency spinner 255

```
Padding Left: 3mm
Padding Right: 3mm
Padding Top: 6mm
Padding Bottom: 3mm
```

- Switch to the Padding tab
- Uncheck the Derive checkbox
- Fill into the spinners the values 3, 3, 6 and 3 in this order
- In all the combo boxes pick Millimeters (approximate)

```
Margin: 0px
```

- Switch to the Margin tab
- Uncheck the Derive checkbox
- Fill into all the spinners the value 0

```
Border: Empty
```

- Switch to the Border tab
- Uncheck the Derive checkbox
- Click the … button on the right hand side. This should open a Border dialog
- Pick Empty in the Type combo box
- Click OK to accept the dialog

- Click OK to add the new entry
- Select File → Save to save your changes

Picking the Right Font

Before we proceed I'd like to take a moment to discuss fonts. Codename One lets you place a TTF font file in the project and work with that. If I was aiming for 100% pixel perfect UI I might have done that but for most cases the native OS font is the best option.

Codename One has several builtin "native:" font families that map to the OS native font for the various platform e.g. on iOS 9+ this will map to the San Francisco font and in older OS's to Helvetica Neue. On Android this will use Roboto and so forth. There are multiple types of "native" font options ranging in weight and style but I mostly use the "light" version (named `native:MainLight` in the designer tool) which closely resembles the Uber font.

Discovering UIID's and Edit In Place

You know that the Toolbar is the UIID for the title area because I told you so. But how would you have discovered that on your own?

For that we need the Component Inspector tool which you can launch from the simulator's Simulate → Component Inspector menu option.

Once launched, you should see the inspector UI and you can gain insight into the layout/theme of the running application.

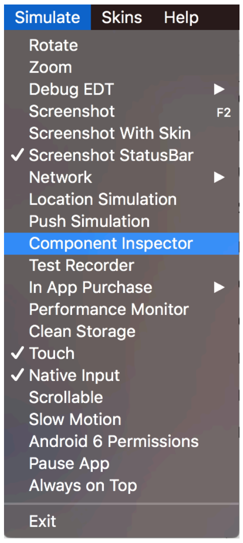
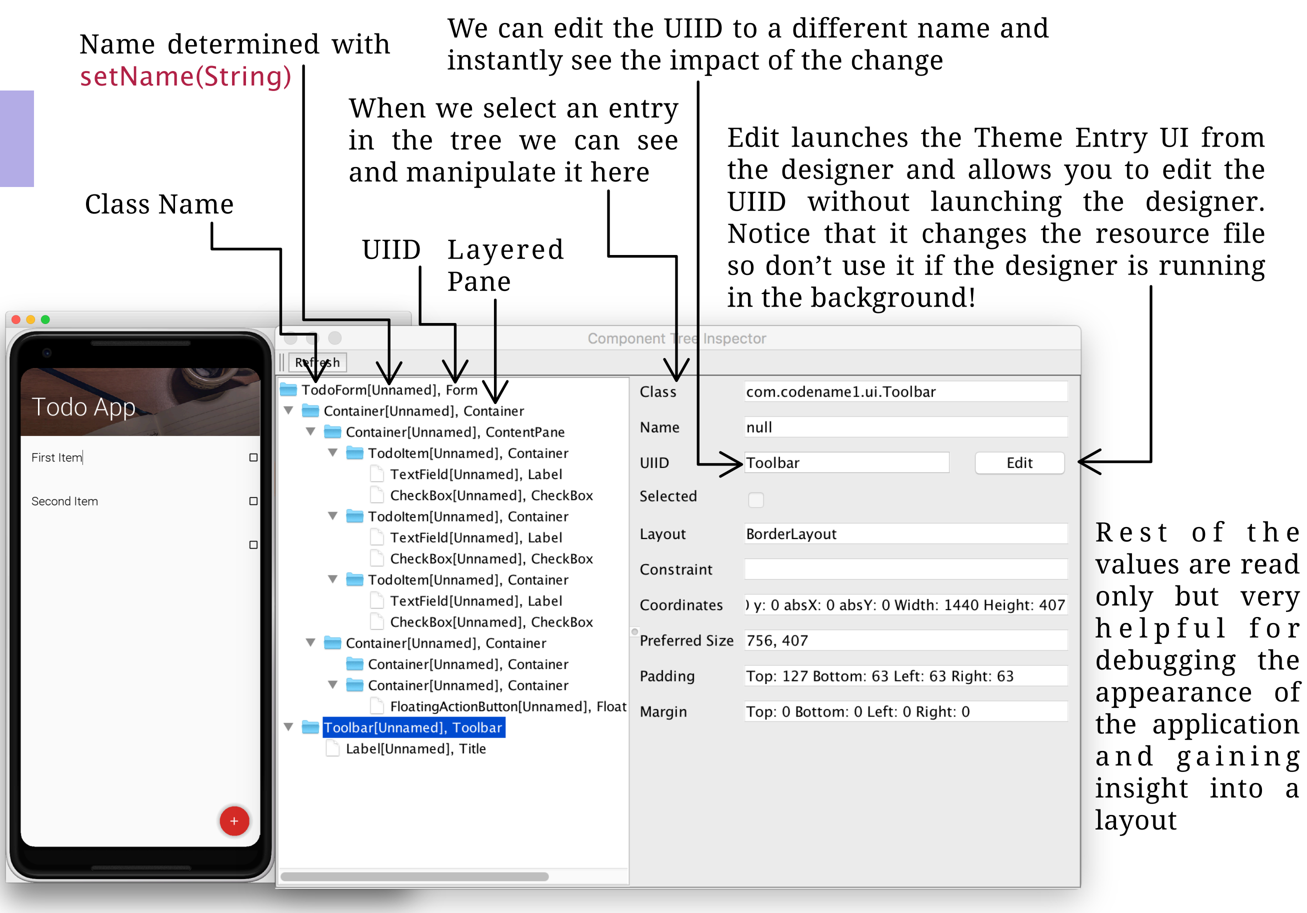


Figure 2. 19. Launching the Component Inspector



Name determined with setName(String)

We can edit the UIID to a different name and instantly see the impact of the change

When we select an entry in the tree we can see and manipulate it here

Class Name

UIID

Layered Pane

Edit launches the Theme Entry UI from the designer and allows you to edit the UIID without launching the designer. Notice that it changes the resource file so don't use it if the designer is running in the background!

Rest of the values are read only but very helpful for debugging the appearance of the application and gaining insight into a layout

```
TodoForm[Unnamed], Form
  Container[Unnamed], Container
    Container[Unnamed], ContentPane
      TodoItem[Unnamed], Container
        TextField[Unnamed], Label
        CheckBox[Unnamed], CheckBox
      TodoItem[Unnamed], Container
        TextField[Unnamed], Label
        CheckBox[Unnamed], CheckBox
      TodoItem[Unnamed], Container
        TextField[Unnamed], Label
        CheckBox[Unnamed], CheckBox
    Container[Unnamed], Container
      Container[Unnamed], Container
      Container[Unnamed], Container
        FloatingActionButton[Unnamed], Float
  Toolbar[Unnamed], Toolbar
    Label[Unnamed], Title
```

Class	com.codename1.ui.Toolbar
Name	null
UIID	Toolbar
Selected	
Layout	BorderLayout
Constraint	
Coordinates	) y: 0 absX: 0 absY: 0 Width: 1440 Height: 407
Preferred Size	756, 407
Padding	Top: 127 Bottom: 63 Left: 63 Right: 63
Margin	Top: 0 Bottom: 0 Left: 0 Right: 0

Figure 2. 20. The Component Inspector Tool

Now that we have a grasp of the tools and we understand how to theme the UI, let's go over the other elements. First we have the Title which I already mentioned before. The Toolbar UIID contains the background image but the Title UIID contains the text of the title.

The Title UIID is **much** simpler:

Listing 2. 19. Title Styling

Foreground Color: 0xffffff ← White text for the title, notice you can just type the hex value into the foreground color field in the designer

Transparency: 0 ← Transparent background means we don't need a background color

Font: native:MainLight 7mm ← That's a relatively thin large font, standard fonts are usually between 2.5mm to 3mm

Once this is done the Todo title is very close to the final result. You can see what we have so far here on the right.

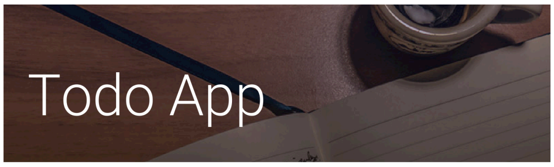


Figure 2. 21. Title on Android Hence the Left Alignment

Notice that the title in iOS would be center aligned because the alignment attribute is derived from the native OS theme

The title looks almost done. Let's move to the design of the body, I'll split the design of the body into two steps to simplify it.

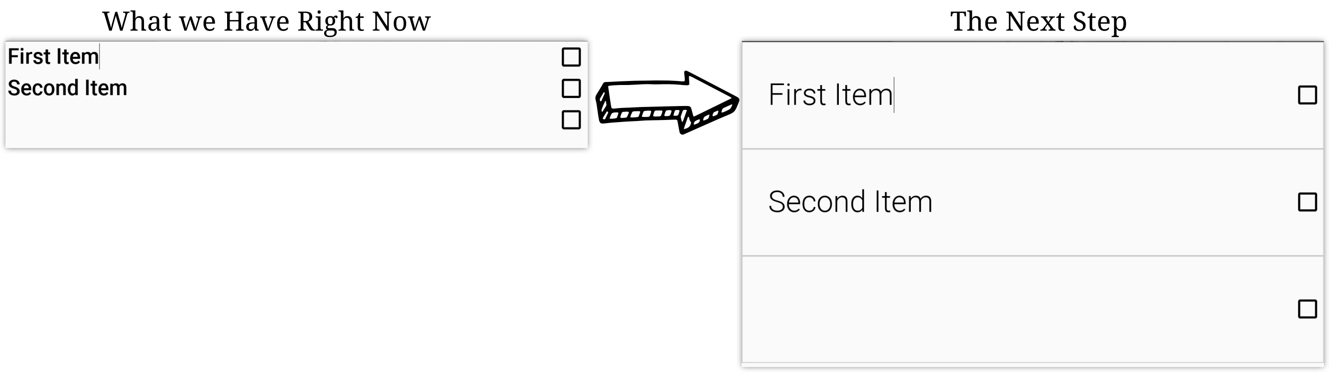


Figure 2. 22. What we have now and the Next Step

Let's start with the obvious. The image on the left is compact and demonstrates clearly the importance of good padding/font choices!

As you recall we styled the text fields with the Label UIID. We can fix the padding and font by using the following style on Label:

Listing 2. 20. Label Styling

```
Padding Left: 3mm
Padding Right: 3mm
Padding Top: 4mm
Padding Bottom: 4mm
Font: native:MainLight 3.5mm
```

Use Derive All in the designer tool to apply this style to the other style modes. When I don't say otherwise do it by default on every Component UIID that can be selected/edited

All this work brought us close to our final destination. The next part of the change would be the lines between the entries. Our gut reaction might be to define an underline for Label. That would work but since the Label doesn't reach the edge of the Form we would have a gap.

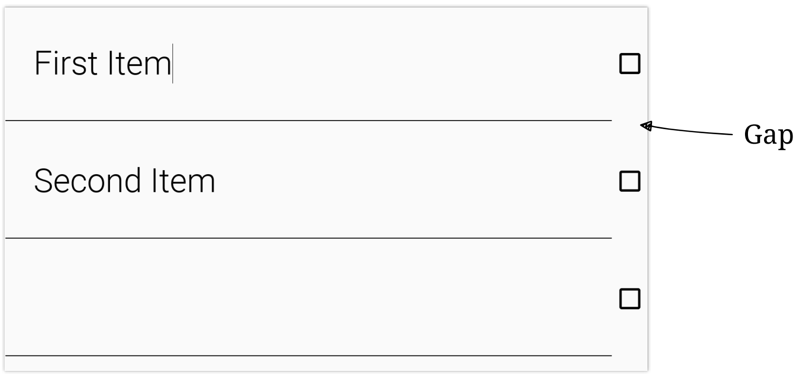


Figure 2. 23. Why Underline on Label isn't what we Want

So what we really want is to underline the whole TodoItem entry. We can do that in code using this code in the TodoItem constructor:

Listing 2. 21. Underline the TodoItem Using Java Code

```
Style s = getAllStyles();
s.setPaddingUnit(Style.UNIT_TYPE_PIXELS);
s.setPadding(0, 2, 0, 0);
s.setBorder(Border.createLineBorder(2, 0xcccccc));
```

Changes made to this object will automatically apply to all 4 style objects

We want to define padding in pixels so we can draw a 2 pixel border

Padding is defined as top/bottom/left/right order. We leave a two pixel padding for the border

We draw a gray underline 2 pixel border

We don't need this code if we use the styling approach through setUIID()

Currently a TodoItem has the Container UIID. When we set the Style object value we change it on an individual object instance. So these changes don't impact other Container instances.

Customizing the Container UIID in the theme is a bad idea. A lot of things rely on the Container UIID and if you change it the impact could be wide

Still I'd rather do things in the theme normally and we can, we just need to change the UIID of the TodoItem class like this:

Listing 2. 22. Underline the TodoItem Using the Theme

```
setUIID ( "Task" );
```

Then I can define the style in the designer tool as such:

Listing 2. 23. Task Styling

```
Padding Left: 0px
Padding Right: 0px
Padding Top: 0px
Padding Bottom: 2px
Border: Underline 0xcccccc 2px
```

And this brings us almost to the last stage of the design changes. We have two more items we'd need to add/change to reach the final design.

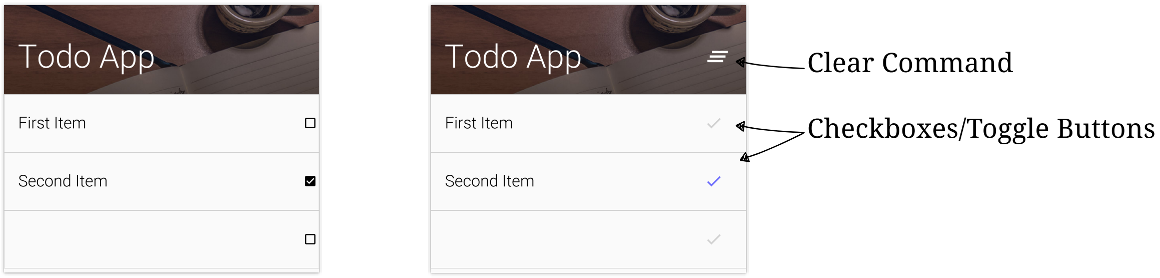


Figure 2. 24. Final UI Changes for the Todo App

These two final changes to the UI combine code and theming. Lets start with the checkboxes. I used the CheckBox class to represent those. I could customize the image of the CheckBox using the theme, but that isn't very flexible. Instead I chose to add these lines to the TodoItem constructor:

Listing 2. 24. CheckBox to Toggle Button in the TodoItem Constructor

```
done.setToggle(true);
FontImage.setMaterialIcon(done,
      FontImage.MATERIAL_CHECK,4);
```

CheckBox can look like a toggle button and effectively hide the default check mark

We set the icon for the checkmark manually from the material icons

Both CheckBox and RadioButton can act as if they are a button and hide the checkmark symbol. This allows a lot of flexibility. However, this also means we need to customize the styling. Once we enable the toggle mode of a Checkbox its UIID changes from CheckBox to ToggleButton.

To make this look like the desired image we need to do the following:

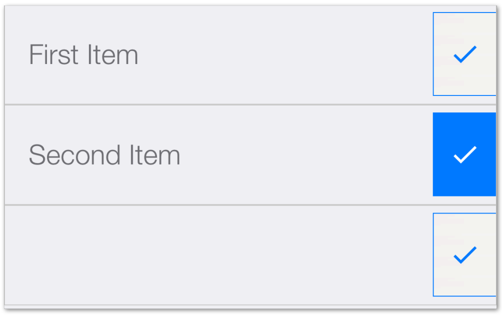


Figure 2. 25. The Current Result Looks Like This

Listing 2. 25. ToggleButton Styling

```
Foreground Color: 0xcfcfcf
Transparency: 0
Border: Empty
```

We don’t want a background color as we’ll use the background of the parent

The border drawn around the button by default should be disabled explicitly

We also need to override the selected and pressed states of the ToggleButton class:

Listing 2. 26. ToggleButton Selected and Pressed Styling

```
Foreground Color: 0x6868FD
Border: Empty
Derive: ToggleButton Unselected
```

We set the foreground when a button is pressed which will make it purple

We define an empty border as the toggle button has a border too

We derive other settings from the unselected version of the component

This makes the toggle checkboxes act like the ones in the finished version and the only thing that remains is the clear command.

The commands fit into the toolbar and are added using syntax such as this:

Listing 2. 27. TodoForm Constructor Adding the Clear Command

```
getToolbar().addMaterialCommandToRightBar("" ,
      FontImage.MATERIAL_CLEAR_ALL, e -> clearAll());
```

Adds a command with a material icon

We pick the icon as clear all and invoke the clearAll method when it's clicked

I'll get to clearAll() soon but I want to finish the styling first.

There are many ways to add commands and they don't have to use the material icons. It's just convenient.

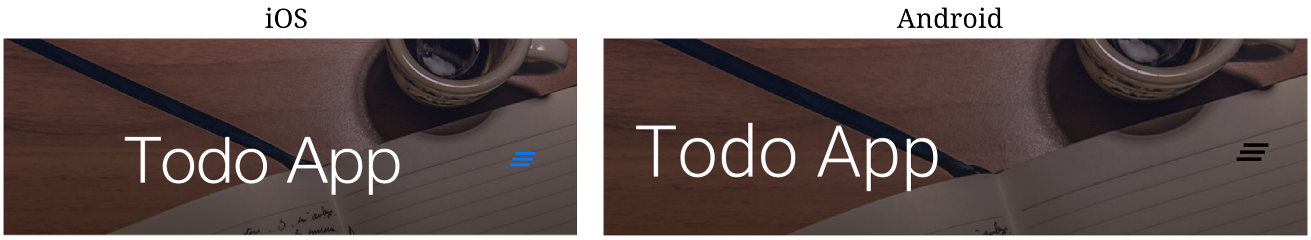


Figure 2. 26. Title Command Before Styling

If you look at the UI before we apply the style you will notice the command image is also much smaller. The FontImage class uses the size of the existing font in the style to define the size of the icon. So if we increase the size of the font in the TitleCommand UIID it should apply to the icon too.

Listing 2. 28. TitleCommand Styling

```
Foreground Color: 0xffffff
Transparency: 0
Font: native:MainLight 5mm
```

The title was styled as 7mm so this is reasonably smaller

With this, the application should now look like it does in figure 2.1 (page 36) and the UI design aspect is complete!

But before we move on to event handling let's cover the clearAll method. In order to do that I'll also need to stub save and load methods which I'll implement later in the persistence section:

Listing 2. 29. TodoForm

```
private void clearAll() {
    int cc = getContentPane().getComponentCount();
    for(int i = cc - 1 ; i >= 0 ; i--) {
        TodoItem t = (TodoItem)getContentPane().getComponentAt(i);
        if(t.isChecked()) {
            t.remove();
        }
    }
    save();
    getContentPane().animateLayout(300);
}
private void load() {}
private void save() {}
```

Clear all removes all of the checked Todo items

We're looping backwards from the end. That means that if we remove a component the offset still won't change

If an item is checked we remove it from its parent (the content pane)

This is a type of `revalidate()` that animates. After the components are removed the remaining components will slide into place for 300ms

We'll implement these later in the chapter, for now a stub will do

And with this, clear will work as well by clearing the checked items from our todo list.

2.2.3 *Event Handling*

Now that we have the UI working lets dig deeper into the functionality and events. I've used events before in the code but skimmed over them e.g. this is code we had for handling the click event on the `FloatingActionButton`:

Listing 2. 30. TodoForm Constructor The FloatingActionButton Event

```
fab.addActionListener(e -> addNewItem());
```

If you are new to Java or haven't used it in a while this code might look weird. It's a lambda expression which was added to Java 8. The equivalent code in Java 5 would be:

Listing 2. 31. The FloatingActionButton Event Without Lambda

```
fab.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        addNewItem();
    }
});
```

Action listener is an interface from com.codename1.ui.events

This is the same addNewItem call we had in the lambda, the rest is boilerplate

You will notice that the lambda expression strips away a lot of the "dead weight code". If we have more than one line of code to write we can still use a lambda expression with curly brackets as such:

Listing 2. 32. The FloatingActionButton Event Lambda with Brackets

```
fab. addActionListener (e -> {
    addNewItem();
});
```

Notice we also had to add a semicolon

Observers and Event Types

The approach of adding a listener is called the observer pattern and it's common for most modern UI frameworks/OS's. We can register an interest in receiving an event and deregister that interest when we no longer need the event (e.g. `removeActionListener`).

We usually don't need to remove a listener to "cleanup". The garbage collector will remove both the component and the listener together when we are done with both

`ActionListener` is the workhorse of events in Codename One but there are also some other similar event types such as `DataChangedListener` which is used to monitor changes to the `TextField` as you type etc.

We can demonstrate event handling in our app by adding the infrastructure for persistence support. You might notice that the UI of the app doesn't include a save button. Mobile apps usually save automatically. This makes sense… You might work on something, get a phone call and forget about it. Saving implicitly makes a lot of sense for a mobile device.

I mentioned the save() method before. I'd like to create event handlers that invoke it whenever data changes. To do this I'll need to first change some things in TodoForm:

Listing 2. 33. Event Changes to TodoForm

```
public class TodoForm extends Form {
    private ActionListener saver;
    public TodoForm() {
        // most of the constructor didn't change
        load();
    }
    private ActionListener getAutoSave() {
        if (saver == null) {
            saver = (e) -> save();
        }
        return saver;
    }
    private void addNewItem() {
        TodoItem td = new TodoItem("", false,
            getAutoSave());
        // rest of the method didn't change
    }
    // rest of the class didn't change
}
```

We will add a load() method soon similar to the save() method

saver is an ActionListener that invokes save()

saver is passed to the TodoItem where it receives change notifications

This implies changes to the TodoItem:

Listing 2. 34. Event Changes in the TodoItem

```
public class TodoItem extends Container {
    public TodoItem(String name, boolean checked, ActionListener onChange) {
        // most of the constructor didn't change
        nameText.addActionListener(onChange);
        done.addActionListener(onChange);
    }
    // rest of the class didn't change
}
```

We use the action listener and bind it directly to the saver call

This code invokes save on any UI change without an explicit action from the user.

Event Dispatch Thread

Codename One is single threaded. All events and almost all method calls occur on a single thread called the Event Dispatch Thread (EDT). By using just one thread Codename One can avoid complex synchronization code and focus on simple functionality that assumes only one thread.

> You can assume that all code will occur on a single thread and avoid complex synchronization logic within your own code

Every call you receive from Codename One will occur on the EDT. E.g. every event, calls to `paint()`, lifecycle calls (`start()` etc.) always occurs on the EDT.

This is pretty powerful, however it means that as long as your code is processing nothing else can happen in Codename One!

> If your code takes too long to execute then no painting or event processing will occur during that time, so a call to `Thread.sleep()` will actually stop everything!
> This is commonly known as "blocking the EDT" and would grind your performance to a halt

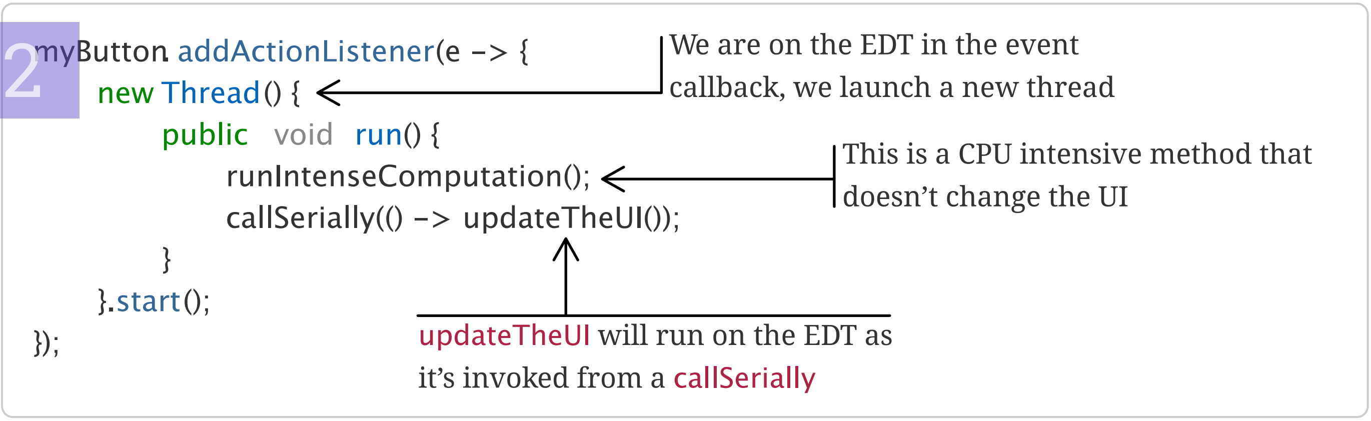
When you need to run a CPU intensive task you should spawn a `Thread` and do the work there. Codename One's networking code automatically spawns its own network thread and performs all networking on separate threads. However, this also poses a problem…

Codename One assumes all modifications to the UI are performed on the EDT but if we spawned a separate thread (or did networking). How do we force our modifications back into the EDT?

For that purpose we have two methods:

- `callSerially(Runnable)` – a thread can invoke `callSerially` to execute the given `Runnable` object on the EDT
- `callSeriallyAndWait(Runnable)` – identical to `callSerially` but it returns when the `Runnable` finished its execution

Listing 2. 35. callSerially Sample



```
myButton.addActionListener(e -> {
    new Thread() {
        public void run() {
            runIntenseComputation();
            callSerially(() -> updateTheUI());
        }
    }.start();
});
```

We are on the EDT in the event callback, we launch a new thread

This is a CPU intensive method that doesn't change the UI

updateTheUI will run on the EDT as it's invoked from a callSerially

This allows us to use a thread for a CPU intensive task and get back into the UI when we are done.

Codename One supports a more elaborate tool called invokeAndBlock which spawns a thread while "legally" blocking the EDT

2.2.4 *IO and Storage*

There are 3 standard storage locations in Codename One:

- **Storage** - This is an OS specific storage location that's closely coupled to the app. It's normally very portable and also simple, things such as directories or paths aren't supported
- **FileSystemStorage** - This is often confused with storage because in some OS's there is an overlap. This is the native OS File System. It provides more capabilities such as directories. The downside is complexity and potential compatibility issues due to device differences. This system always expects a full file path
- **SQLite** - The standard SQL database built into iOS, Android and Windows devices

I'll only focus on Storage right now as we won't use the others in this application. Lets look at the implementation of save() and load().

Listing 2. 36. Save and Load in TodoForm

```
private void save() {
    try(DataOutputStream dos = new DataOutputStream(
            createStorageOutputStream("todo-list-of-items"))) {
        dos.writeInt(
            getContentPane().getComponentCount());
        for(Component c : getContentPane()) {
            TodoItem i = (TodoItem)c;
            dos.writeBoolean(i.isChecked());
            dos.writeUTF(i.getText());
        }
    } catch(IOException err) {
        Log.e(err);
        ToastBar.showErrorMessage("Error saving todo list!");
    }
}
private void load() {
    if(existsInStorage("todo-list-of-items")) {
        try(DataInputStream dis = new DataInputStream(
                createStorageInputStream("todo-list-of-items"));) {
            int size = dis.readInt();
            for(int iter = 0 ; iter < size ; iter++) {
                boolean checked = dis.readBoolean();
                TodoItem i =
                    new TodoItem(dis.readUTF (), checked, getAutoSave();
                add(i);
            }
        } catch (IOException err) {
            Log.e(err);
            ToastBar.showErrorMessage( "Error loading todo list!");
        }
    }
}
```

This opens a storage file for writing with the given name

DataOutputStream provides convenient methods like writeInt and writeUTF

We write the values of every component in binary form

Exceptions aren't likely for this case but if they happen we log them and show an error message using the ToastBar

The first time we run the input file won't exist

Notice that this is the exact inverse of the write method

The ToastBar class presents a small notification typically at the bottom of the Form. These notifications expire by default after a few seconds

I could have used the JSON parser or some other tool for writing/reading the data but I chose to do something simple right now.

We can now run the Todo App and it will remember everything we add and change within the application. The Todo app is now complete!

2.2.3 *Summary*

In this chapter, we learned:

- How to manage layout and scrolling behavior so we can build complex component hierarchies
- Styling components using the designer tool and UIID's to create elaborate looks for our applications
- How to use background threads with Codename One and go back and forth to the main event dispatch thread. We can thus create more performant applications by leveraging the CPU more effectively
- How to save and load information from persistent storage so our application can retain data between executions

After this chapter you should have enough understanding of Codename One to get you through the book. I'll still take detours along the way to explain some things I didn't get to in these first two chapters but I'm anxious to dive into the Uber app as I'm sure you are!

There is still one small chapter and then we can get started…

Appendix A: Setup Codename One

A

This section is divided into the three separate IDE’s supported by Codename One: IntelliJ/IDEA, NetBeans and Eclipse.

Before you begin make sure to install JDK 8. We recommend the version from Oracle. If you install OpenJDK make sure to install JavaFX support as well.

> ***As of this writing JDK 9 isn’t supported yet!***
> Support for newer JDK’s is planned for Codename One 5.0 so this might change by the time you read this. Check out www.codenameone.com/download.html for the current status

Since the IDE runs on top of a JDK instance we recommend running the IDE itself on JDK 8 to avoid problems.

> The screenshots are from Mac OS but the process should work exactly the same on Windows and Linux

A.1. IntelliJ/IDEA

A.1

Codename One recommends IntelliJ/IDEA 2016 or newer.

> Codename One **doesn’t support** Android Studio! You can use IntelliJ/IDEA community edition instead

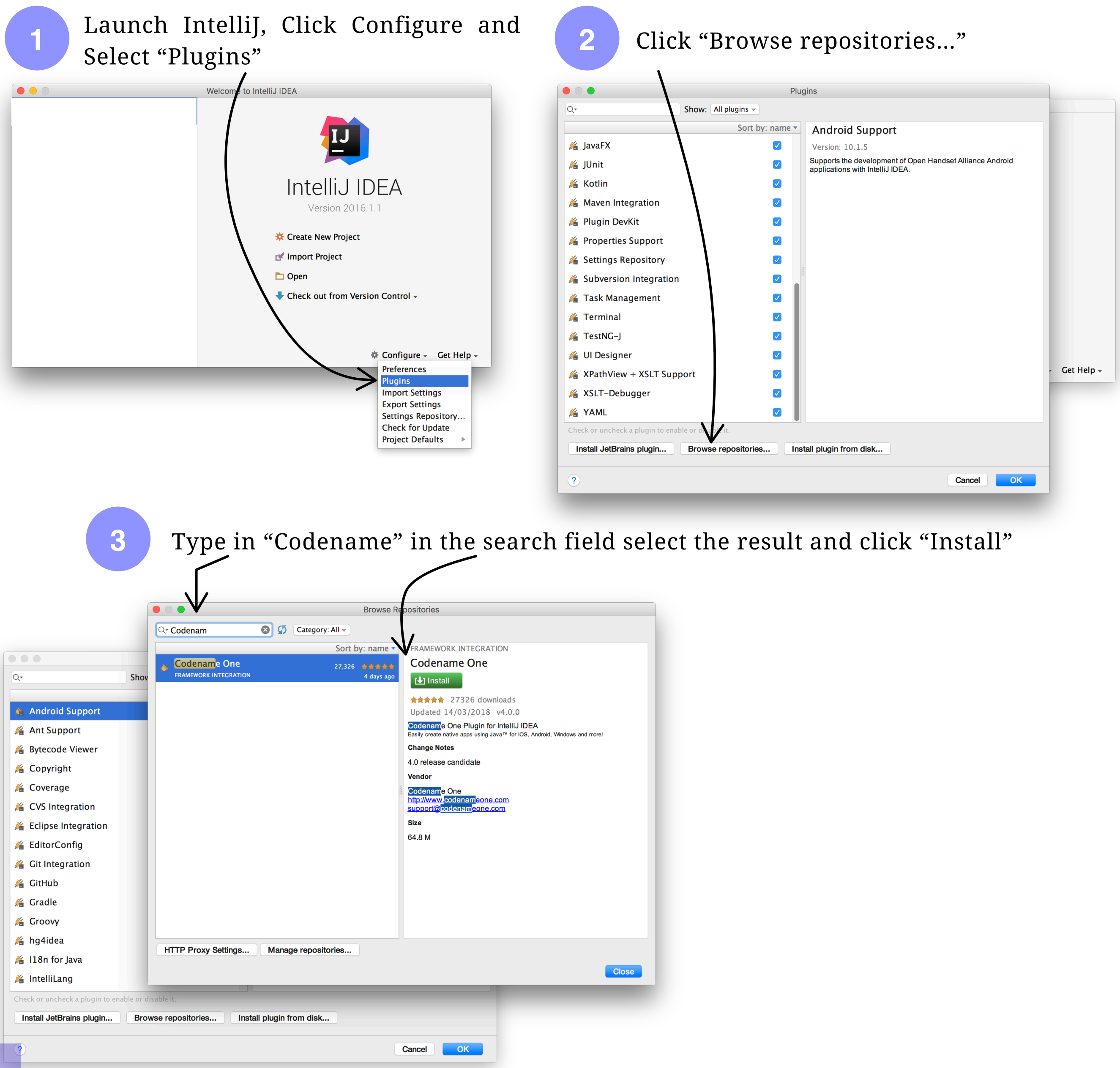
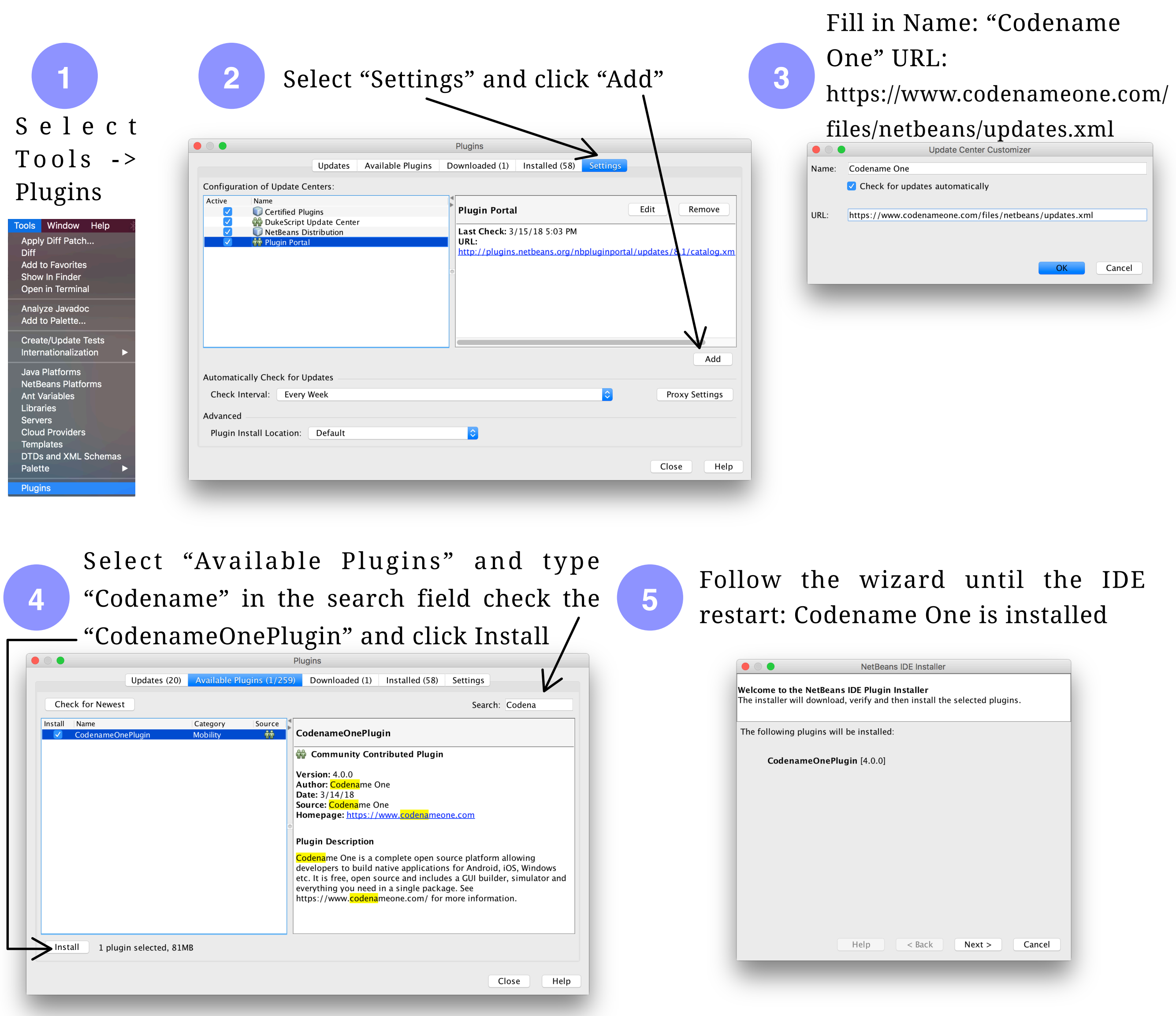


Figure 14. 1. IntelliJ Installation Instructions

A.2 NetBeans

NetBeans install is pretty simple although the default “plugin center” for NetBeans is notoriously unreliable. That’s why we recommend using the Codename One plugin center: www.codenameone.com/files/netbeans/updates.xml



1 Select Tools -> Plugins

2 Select “Settings” and click “Add”

3 Fill in Name: “Codename One” URL: https://www.codenameone.com/files/netbeans/updates.xml

4 Select “Available Plugins” and type “Codename” in the search field check the “CodenameOnePlugin” and click Install

5 Follow the wizard until the IDE restart: Codename One is installed

Plugins

Configuration of Update Centers:

Active	Name
✓	Certified Plugins
✓	DukeScript Update Center
✓	NetBeans Distribution
✓	Plugin Portal

Plugin Portal

Last Check: 3/15/18 5:03 PM

URL: http://plugins.netbeans.org/nbpluginportal/updates/8.1/catalog.xml

Automatically Check for Updates

Check Interval: Every Week

Advanced

Plugin Install Location: Default

Update Center Customizer

Name: Codename One

✓ Check for updates automatically

URL: https://www.codenameone.com/files/netbeans/updates.xml

Plugins

Search: Codena

Install	Name	Category	Source
✓	CodenameOnePlugin	Mobility	

CodenameOnePlugin

Community Contributed Plugin

Version: 4.0.0
Author: Codename One
Date: 3/14/18
Source: Codename One
Homepage: https://www.codenameone.com

Plugin Description

Codename One is a complete open source platform allowing developers to build native applications for Android, iOS, Windows etc. It is free, open source and includes a GUI builder, simulator and everything you need in a single package. See https://www.codenameone.com/ for more information.

1 plugin selected, 81MB

NetBeans IDE Installer

Welcome to the NetBeans IDE Plugin Installer
The installer will download, verify and then install the selected plugins.

The following plugins will be installed:

CodenameOnePlugin [4.0.0]

Figure 14. 2. NetBeans Installation Instructions

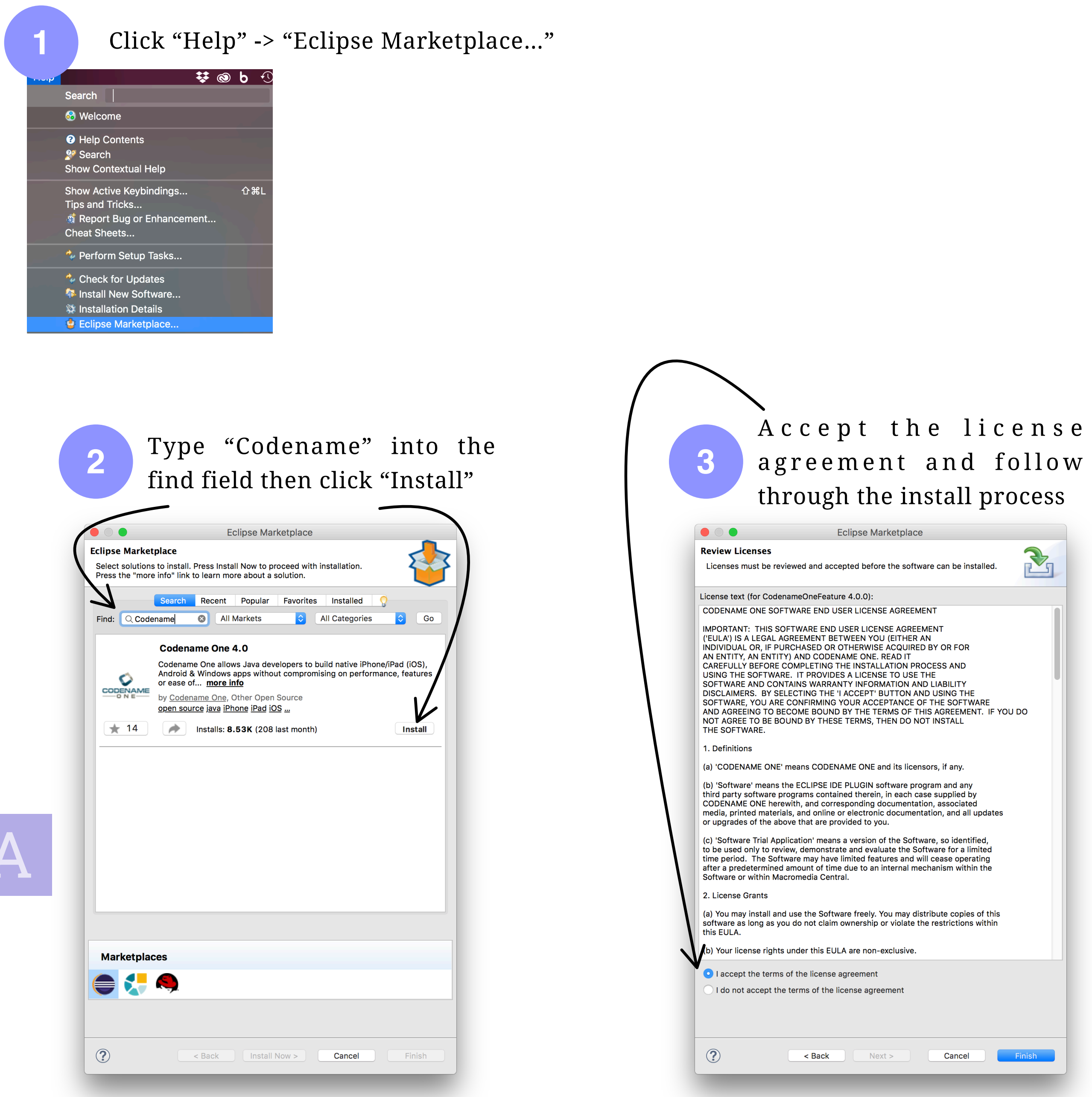
Make sure you are using a NetBeans version that includes Java support, don’t download a version for Ruby/PHP or J2ME and make sure the IDE runs on top of JDK 8

Eclipse

Codename One supports Eclipse Neon 2 or newer. There are a few pitfalls that can happen with an Eclipse install specifically when other JVM versions are installed on your machine.

If you are new to Java, Eclipse might be intimidating. It’s a very powerful IDE but its configuration is rough

Make sure your JAVA_HOME environment variable points at JDK 8 and that the path to the JDK 8 bin directory is first in the PATH statement. If all else fails edit the eclipse.ini file to force Eclipse to use your JDK 8 install. See this site for help with editing the eclipse.ini file: wiki.eclipse.org/Eclipse.ini


1 Click “Help” -> “Eclipse Marketplace…”

2 Type “Codename” into the find field then click “Install”

Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Find: Codename | All Markets | All Categories | Go

Codename One 4.0

Codename One allows Java developers to build native iPhone/iPad (iOS), Android & Windows apps without compromising on performance, features or ease of... more info

by Codename One, Other Open Source
open source java iPhone iPad iOS ...

14 Installs: **8.53K** (208 last month) Install

3 Accept the license agreement and follow through the install process

Eclipse Marketplace

Review Licenses

Licenses must be reviewed and accepted before the software can be installed.

License text (for CodenameOneFeature 4.0.0):

CODENAME ONE SOFTWARE END USER LICENSE AGREEMENT

IMPORTANT: THIS SOFTWARE END USER LICENSE AGREEMENT ('EULA') IS A LEGAL AGREEMENT BETWEEN YOU (EITHER AN INDIVIDUAL OR, IF PURCHASED OR OTHERWISE ACQUIRED BY OR FOR AN ENTITY, AN ENTITY) AND CODENAME ONE. READ IT CAREFULLY BEFORE COMPLETING THE INSTALLATION PROCESS AND USING THE SOFTWARE. IT PROVIDES A LICENSE TO USE THE SOFTWARE AND CONTAINS WARRANTY INFORMATION AND LIABILITY DISCLAIMERS. BY SELECTING THE 'I ACCEPT' BUTTON AND USING THE SOFTWARE, YOU ARE CONFIRMING YOUR ACCEPTANCE OF THE SOFTWARE AND AGREEING TO BECOME BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO BE BOUND BY THESE TERMS, THEN DO NOT INSTALL THE SOFTWARE.

1. Definitions

(a) 'CODENAME ONE' means CODENAME ONE and its licensors, if any.

(b) 'Software' means the ECLIPSE IDE PLUGIN software program and any third party software programs contained therein, in each case supplied by CODENAME ONE herewith, and corresponding documentation, associated media, printed materials, and online or electronic documentation, and all updates or upgrades of the above that are provided to you.

(c) 'Software Trial Application' means a version of the Software, so identified, to be used only to review, demonstrate and evaluate the Software for a limited time period. The Software may have limited features and will cease operating after a predetermined amount of time due to an internal mechanism within the Software or within Macromedia Central.

2. License Grants

(a) You may install and use the Software freely. You may distribute copies of this software as long as you do not claim ownership or violate the restrictions within this EULA.

(b) Your license rights under this EULA are non-exclusive.

[✓ I accept the terms of the license agreement]
I do not accept the terms of the license agreement

Figure 14. 3. Eclipse Installation Instructions

In order to run the app in Eclipse make sure to select the .launch file in Eclipse

Appendix B: Setup Spring Boot and MySQL

B

This chapter covers:

- How to setup MySQL/MariaDB and map it to Spring Boot
- How to setup a Spring Boot with the Maven build process

We use Spring Boot for the server and use Maven to build it. When I developed the code for the book 1.5.7 was the latest stable version of Spring Boot so I stuck to that. However, the followup Facebook clone application worked with 2.0 without a problem and only required minor adjustments so this should be reasonably easy to migrate if you choose to do so.

All 3 major Java IDE's have Maven plugins or builtin support, so working with Maven should be simple.

MySQL Setup B.1

We begin by installing MySQL or MariaDB on the development machine. I use MySQL during development since it has an easy to use Mac OS installer. However, in production of apps on Linux I tend to prefer MariaDB which is compatible with MySQL. Both should be practically interchangeable as MariaDB is a fork of MySQL.

Installation of both databases is trivial, Oracle provides a free community edition of MySQL here: dev.mysql.com/downloads/mysql/

You can download MariaDB from: downloads.mariadb.org/

Both sites include detailed setup instructions that you should follow.

Once installed you can launch the MySQL command prompt:

Listing 15. 1. Launch MySQL Unix/Linux

```
/usr/local/mysql/bin/mysql -h localhost -u root -p
```

The syntax is identical on Windows except for the path to the MySQL executable.

You need to provide the password given to you during the install process when the app prompts you. At this point you should have a MySQL prompt.

> ***A Simpler Way***
>
> You can use a visual tool such as NetBeans or Toad to connect to the MySQL database and manage it

In the prompt create the new Uber database:

Listing 15. 2. Create Database

```
CREATE DATABASE uberapp;
```

It's possible the database will make you set the password the first time around. You can do it with this code:

Listing 15. 3. Set new Password

```
Password=PASSWORD('your_new_password') WHERE User='root';
```

B.2 Setup Spring Boot Project

One of the best ways to start with Spring Boot is through one of the IDE plugins that offer instant setup wizards. At the time of this writing I found these IDE plugins but check your IDE for updated developments:

- NetBeans - plugins.netbeans.org/plugin/67888/nb-springboot
- IntelliJ - www.jetbrains.com/help/idea/2016.3/creating-spring-boot-projects.html
- Eclipse - spring.io/tools/sts/all

You can use the Spring Boot Initializer to generate a project with the following options:

- Cloud Security
- Web Services
- Websocket
- JPA
- MySQL
- Jersey

Alternatively you can just use this maven project which does the same thing and automatically fetches the dependencies:

Listing 15. 4. Maven Spring Boot pom.xml build file

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                    http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.codename1.uberclone</groupId>
   <artifactId>UberClone</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>UberClone</name>
   <description>Uber style application</description>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.7.RELEASE</version>
      <relativePath/>
      <!-- lookup parent from repository -->
   </parent>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8
      </project.reporting.outputEncoding>
      <java.version>1.8</java.version>
   </properties>
   <dependencies>
```

This is standard Maven boilerplate project header with no real data

The following couple of lines include descriptive strings about the app. Since this is internal to the server they don't really matter

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jersey</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
          <groupId>com.braintreepayments.gateway</groupId>
          <artifactId>braintree-java</artifactId>
          <version>2.71.0</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
```

These are the modules we need for this book

```
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                    <executable>true</executable>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

This packages Spring Boot as an executable JAR we can run on the server very easily

The basic main class for a Spring Boot application is trivial. If you used the Spring Boot initializer then the main class is created for you. If not you can use this code:

Listing 15. 5. Main Source file for Spring Boot app

```
@SpringBootApplication
public class UberCloneApplication {
    public static void main(String[] args) {
        SpringApplication.run(UberCloneApplication.class, args);
    }
}
```

You will notice there isn't much here. Everything else is automatically wired

We do need to configure the properties file for Spring specifically:

B

Listing 15. 6. application.properties file

```
spring.datasource.url=jdbc:mysql://localhost/uberapp
spring.datasource.username=root
spring.datasource.password=DatabasePassword
```

This is the URL for the database. Here it's hosted on the same machine but you can point at a different machine. You can provide a different database name as well

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true
```

These 2 lines indicate that we want the database created and updated automatically based on our Java JPA objects

```
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

This driver and following syntax declaration work for MariaDB too

```
spring.http.multipart.max-file-size=800KB
spring.http.multipart.max-request-size=2048KB
```

We define the maximum file size for file upload to the server. Limits are important to prevent an attacker from uploading huge files and crashing our servers

Once we do this we should have Spring Boot and it should map seamlessly to our locally installed SQL databases.

Appendix C: Styling Codename One with CSS

C

To enable CSS support in Codename One you need to flip a switch in Codename One Settings.

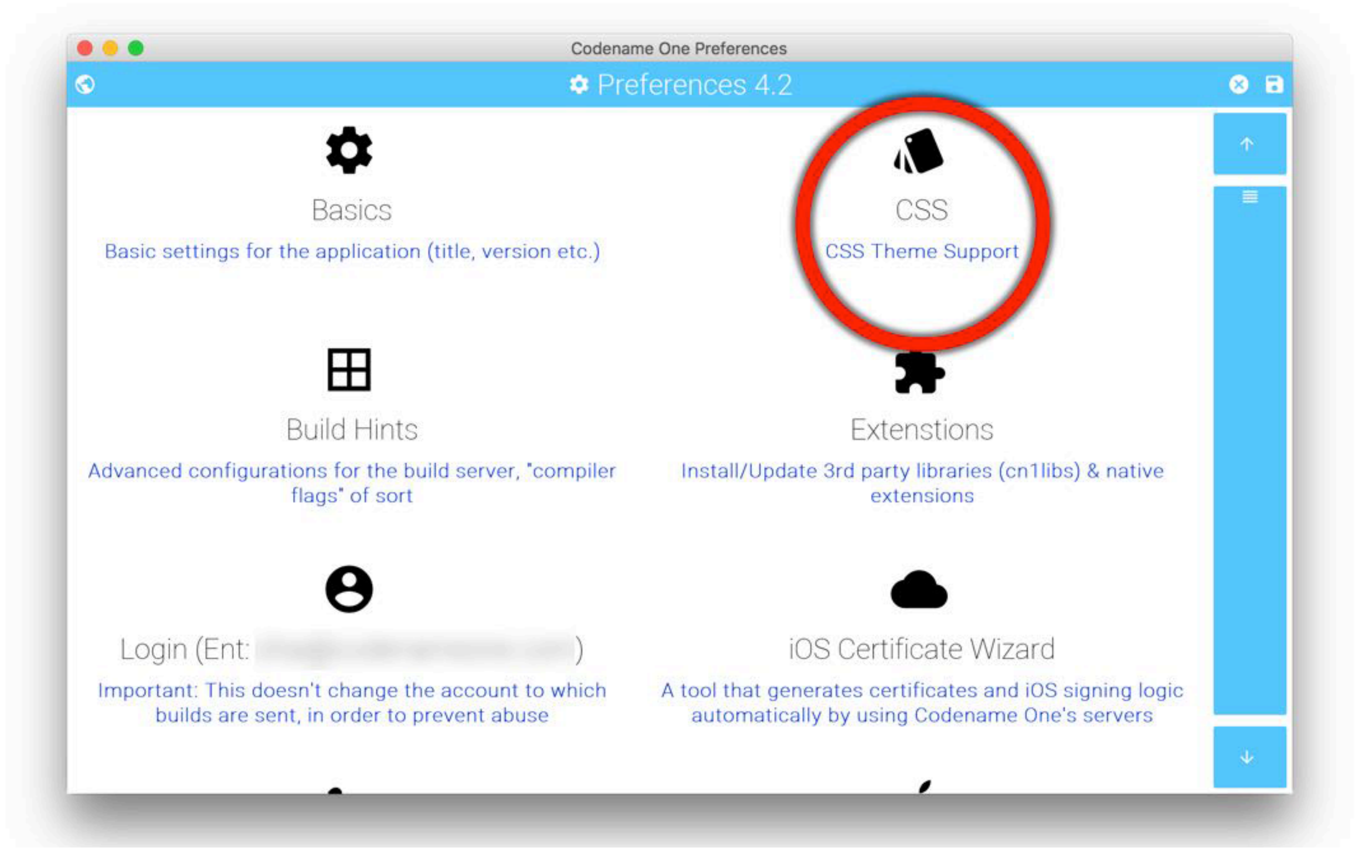

Codename One Preferences
Preferences 4.2
Basics
Basic settings for the application (title, version etc.)
CSS
CSS Theme Support
Build Hints
Advanced configurations for the build server, "compiler flags" of sort
Extenstions
Install/Update 3rd party libraries (cn1libs) & native extensions
Login (Ent:)
Important: This doesn't change the account to which builds are sent, in order to prevent abuse
iOS Certificate Wizard
A tool that generates certificates and iOS signing logic automatically by using Codename One's servers

Figure 16. 1. The CSS Option in Codename One Settings Part I

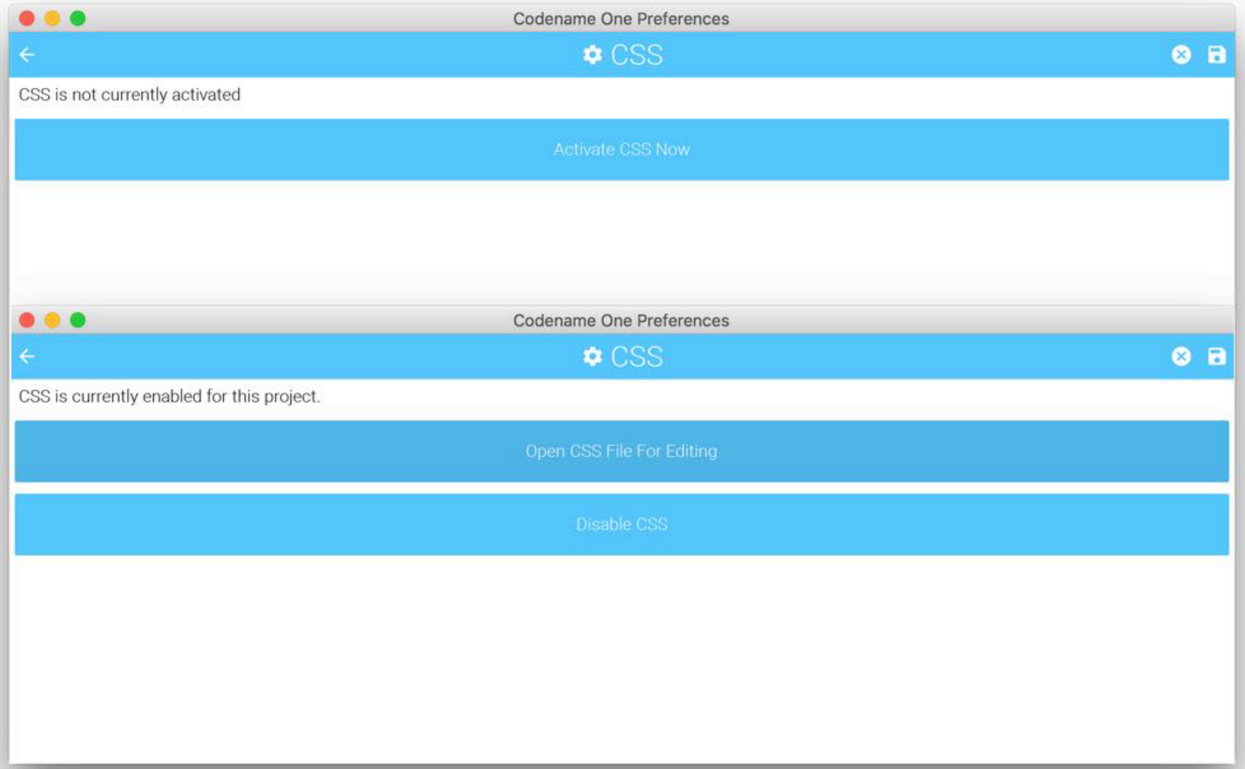

Codename One Preferences
CSS
CSS is not currently activated
Activate CSS Now
Codename One Preferences
CSS
CSS is currently enabled for this project.
Open CSS File For Editing
Disable CSS

Figure 16. 2. The CSS Option in Codename One Settings Part II

Once enabled your `theme.res` file will regenerate from a CSS file that resides under the `css` directory. Changes you make to the CSS file will instantly update the simulator as you save. However, there are some limits to this live update so in some cases a simulator restart would be necessary.

> This appendix assumes you are familiar with CSS and typical CSS terms such as selectors

C.1 Getting Started with CSS

We can now add css files into the `css` directory. If we add the file `css/theme.css` when we compile the project, it will generate the `src/theme.res` file. So changes you make to the resource file in the designer tool will get overwritten!

A most basic hello CSS file would look like this:

Listing 16. 1. Hello CSS

```
#Constants {
    includeNativeBool: true;
}

Label {
    color: blue;
}
```

Theme constants can be defined in this special selector

This is a crucial constant. Otherwise the native theme won’t load

The selector matches the UIID so this is the same as defining the `Label` UIID

Standard CSS attributes should work as expected, color matches foreground and the typical constants work (e.g. blue)

Selectors

All CSS selectors are effectively UIID’s. There is no support for most of the complex selectors e.g. you can’t do something like this:

Listing 16. 2. Nesting Doesn’t Work

```
ContentPane Button {
    /* ... */
}
```

Since CSS is statically compiled it can’t support features that don’t exist in Codename One and complex selectors aren’t supported.

You can however override a specific state of the selector using the suffixes: `.pressed`, `.selected`, `.unselected` or `.disabled`.

Listing 16. 3. Set the Button Pressed Styling

```
Button.pressed {
    /* ... */
}
```

The default targets all states

You can also select multiple targets at once:

Listing 16. 4. Applying Styling to Multiple Types

```
Button.selected, TextField, MyComponent {
    /* ... */
}
```

There are a few extra features I didn't mention which you can read about here: github.com/shannah/cn1-css/wiki/Supported-CSS-Selectors

C.1.2 Properties

The following table lists the main supported properties and notes related to them. For a full list and more details check out github.com/shannah/cn1-css/wiki/Supported-Properties

Table 16. 3. Main Supported Properties

Property	Notes
padding	
margin	
border	Supports the border property and most of its variants (e.g. border-width, border-style, and border-color. It will try to use native CN1 styles for generating borders if possible. If the border definition is too complex, it will fall-back to generating a 9-piece image border at compile-time.
border-radius	
background	
background-color	
background-repeat	
background-image	
font	For more about fonts check out github.com/shannah/cn1-css/wiki/Fonts
font-family	`font-family: "native:MainLight";`

Property	Notes
font-style	
font-size	font-size: 3mm;
color	Foreground color
text-align	
text-decoration	One of: underline , overline , line-through , none, cn1-3d, cn1-3d-lowered, cn1-3d-shadow-north
opacity	

Most of the entries include their respective variants e.g. both margin-top: and margin: 1px 1px 1px 1px; would work.

Images

C.1.3

To add images to the resource file you can place them in the css folder commonly under the images folder within.

Listing 16. 5. Simple Image Usage in CSS

```
SomeStyle {
    background-image: url(images/my-image.png);
    cn1-source-dpi: 480;
}
```

This special property defines the source DPI of the image

This effectively creates a Multi Image in the resource file and automatically scales the image to all the various resolutions. 480 is effectively an HD DPI image.

We can also generate 9-patch borders using images e.g.:

Listing 16. 6. Cutting a 9-patch Border in CSS

```
MyStyle {
    background-image: url(myimage.png);
    cn1-9patch: 5px 8px 4px 10px;
}
```

The distance for cutting from the edge

C

Summary

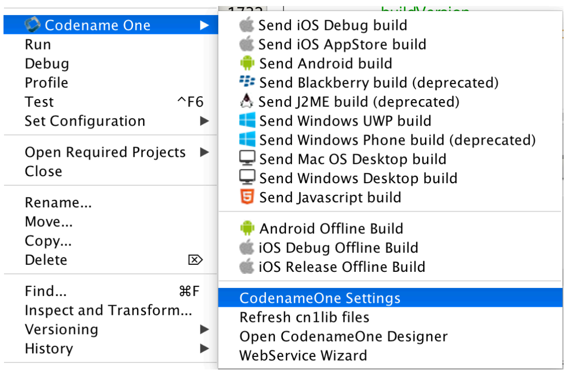
I just barely scratched the surface of the CSS functionality in Codename One. We plan to include thorough coverage of CSS in the Codename One Developer guide version 5.0.

Appendix D: Installing cn1libs

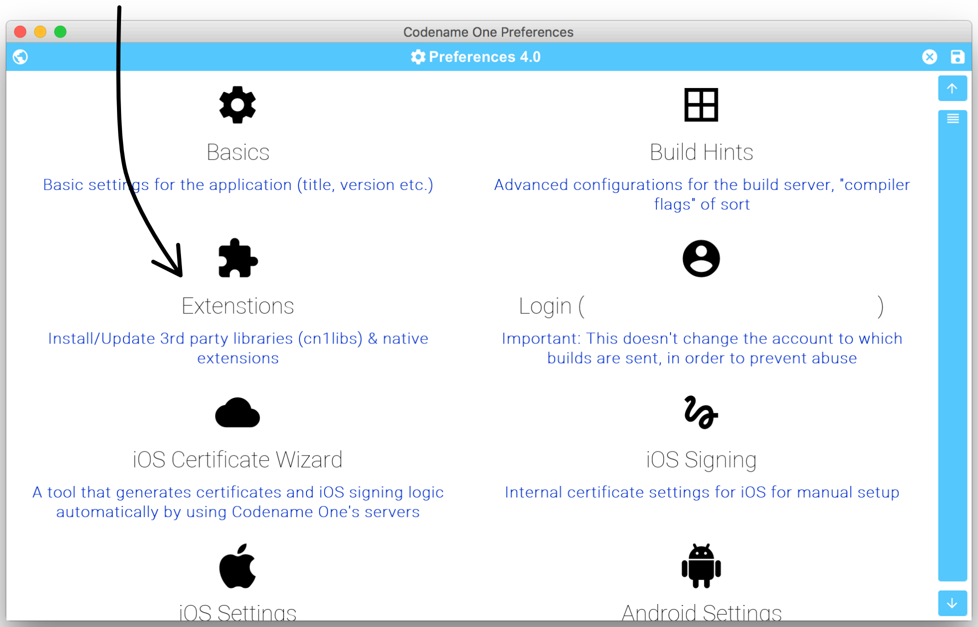
D

Installing a cn1lib is a relatively easy, as illustrated here:

1 Right click the project and select “Codename One” -> “Codename One Settings”



2 Click “Extensions”



Basic settings for the application (title, version etc.)

Advanced configurations for the build server, "compiler flags" of sort

Install/Update 3rd party libraries (cn1libs) & native extensions

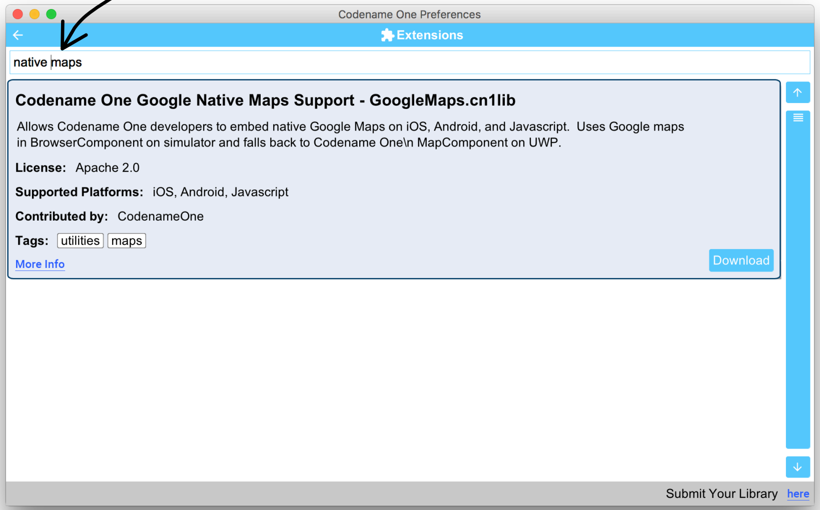
Important: This doesn't change the account to which builds are sent, in order to prevent abuse

A tool that generates certificates and iOS signing logic automatically by using Codename One's servers

Internal certificate settings for iOS for manual setup

Figure D. 1. Overview of the cn1lib Install Process 1 & 2

3 Type the name of the extension you are looking for in the search field and click the Download Button


Codename One Preferences

Extensions

native maps

Codename One Google Native Maps Support - GoogleMaps.cn1lib

Allows Codename One developers to embed native Google Maps on iOS, Android, and Javascript. Uses Google maps in BrowserComponent on simulator and falls back to Codename One\n MapComponent on UWP.

License: Apache 2.0

Supported Platforms: iOS, Android, Javascript

Contributed by: CodenameOne

Tags: utilities maps

More Info

Download

Submit Your Library here

4 Close Settings. Right-click the project and select “Codename One” -> “Refresh cn1lib files”

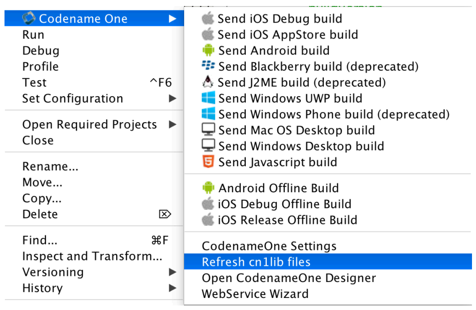
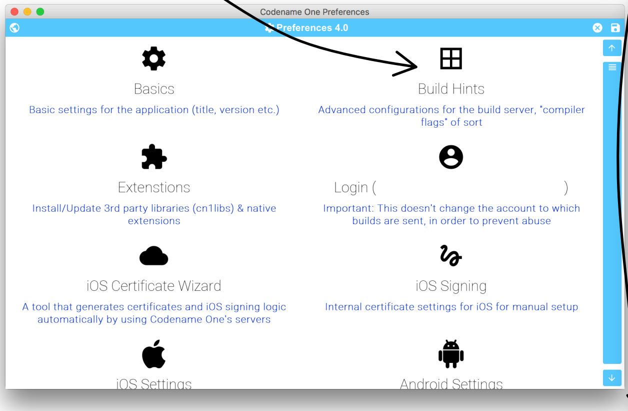


Figure D. 2. Overview of the cn1lib Install Process 3 & 4

5 You might be done already. If your cn1lib needs custom build hints (e.g. Google Maps) relaunch Codename One Settings and click “Build Hints”

6 You can edit the values in the build hints. Specifically, the google maps requires adding entries like android.xapplication which you can add by pressing the “Add Hint” button below


Codename One Preferences

Preferences 4.0

Basics
Basic settings for the application (title, version etc.)

Build Hints
Advanced configurations for the build server, "compiler flags" of sort

Extenstions
Install/Update 3rd party libraries (cn1libs) & native extensions

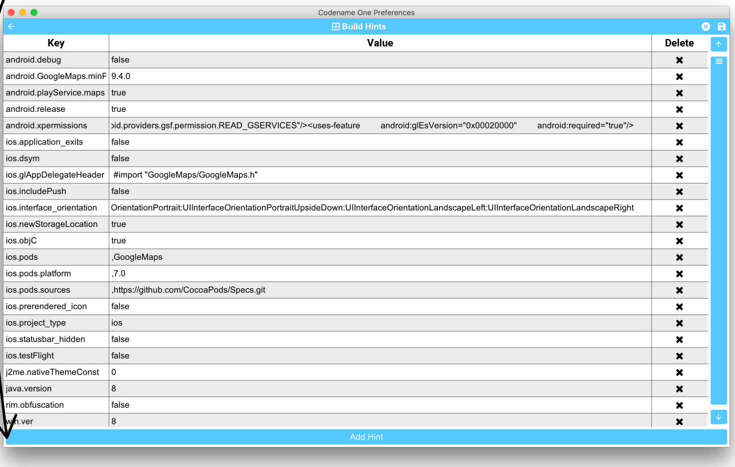
Login ( )
Important: This doesn't change the account to which builds are sent, in order to prevent abuse

iOS Certificate Wizard
A tool that generates certificates and iOS signing logic automatically by using Codename One's servers

iOS Signing
Internal certificate settings for iOS for manual setup

iOS Settings

Android Settings


Codename One Preferences

Build Hints

Key	Value	Delete
android.debug	false	✖
android.GoogleMaps.minF	9.4.0	✖
android.playService.maps	true	✖
android.release	true	✖
android.xpermissions	oid.providers.gsf.permission.READ_GSERVICES"/><uses-feature android:glEsVersion="0x00020000" android:required="true"/>	✖
ios.application_exits	false	✖
ios.dsym	false	✖
ios.glAppDelegateHeader	#import "GoogleMaps/GoogleMaps.h"	✖
ios.includePush	false	✖
ios.interface_orientation	OrientationPortrait:UIInterfaceOrientationPortraitUpsideDown:UIInterfaceOrientationLandscapeLeft:UIInterfaceOrientationLandscapeRight	✖
ios.newStorageLocation	true	✖
ios.objC	true	✖
ios.pods	,GoogleMaps	✖
ios.pods.platform	,7.0	✖
ios.pods.sources	,https://github.com/CocoaPods/Specs.git	✖
ios.prerendered_icon	false	✖
ios.project_type	ios	✖
ios.statusbar_hidden	false	✖
ios.testFlight	false	✖
j2me.nativeThemeConst	0	✖
java.version	8	✖
rim.obfuscation	false	✖
win.ver	8	✖

Add Hint

Figure D. 3. Overview of the cn1lib Install Process 5 & 6

Installing a cn1lib is usually seamless:

- Launch Codename One Settings
- Click Extensions
- Select the extension which is downloaded for you (you can type in the search box)
- Select Refresh Cn1Libs in the right click menu

There are a few cn1libs that require additional configuration and the native Google Maps is one of those cn1libs as it requires several build hints that can’t be picked in runtime. Specifically the build hints are:

```
javascript.googlemaps.key=YOUR_JAVASCRIPT_API_KEY android.xapplication=<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="YOUR_ANDROID_API_KEY"/> ios.afterFinishLaunching=[GMSServices
    provideAPIKey:@"YOUR_IOS_API_KEY"];
```

The reason these are needed is due to the key values that must be present during build time. The key values are values you need to retrieve from the Google Cloud Console as explained here: github.com/codenameone/codenameone-google-maps/

D

Appendix E: Push Notification

E

A huge part of the driver app is the push notification process, that's how we notify a driver that there is a ride pending. But what is push and why should we use it in this case?

Push notification allows us to send a notification to a device while the application might be in the background. This is important both as a marketing tool and as a basic communications device. In this case the driver might be in a different app but we still want him to notice that we are looking for a ride…

Why Push and Not Polling/WebSocket?

Polling the server (periodically asking the server for an update) seem like sensible time proven strategy. However, there are many complexities related to that approach in mobile phones.

The biggest problem is that a polling application will be killed by the OS as it is sent to the background to conserve OS resources. While this might work in some OS's and some cases this isn't something you can rely on. E.g. Android 6+ tightened the background process behavior significantly.

The other issue is battery life, new OS's expose battery wasting applications and as a result might trigger uninstalls. This makes even foreground polling less appealing.

What Is Push? E.1

If you are new to mobile development then you might have heard a lot of buzzwords and very little substance. The problem is that iOS and Android have very different ideas of what push is and should be. For Android, push is a communication system that the server can initiate. E.g. the cloud can send any packet of data, and the device can process it in rather elaborate ways.

For iOS push is mostly a visual notification triggered by the server to draw attention to new information inside an app. These don't sound very different until you realize that in Android you can receive/process a push without the awareness of the end user. In iOS a push notification is displayed to the user, but the app might be unaware of it!

Background Push in iOS is Different

iOS will only deliver the push notification to the app, if it is running or if the user clicked the push notification popup!

Codename One tried to make both OS's "feel" similar so background push calls act the same in iOS and Android as a result.

Push isn't 100% Reliable

You shouldn't push important data. Push is lossy and shouldn't include a payload that **MUST** arrive!
Instead, use push as a flag to indicate that the server has additional data for the app to fetch

For this case we use push to let the drivers know, but pass the actual important information within the socket connection.

E.2 *Various Types of Push Messages*

In the driver app we send a push type 3 message which might have been a bit unclear. Before we proceed I think it's a good time to discuss the various types of push messages.

- `0`, `1` – The default push types. They work everywhere and present the string as the push alert to the user
- `2` – hidden, non-visual push. This won't show any visual indicator on any OS!
  In Android this will trigger the push(String) call with the message body. In iOS this will only happen if the application is in the foreground otherwise the push will be lost
- `3` – allows combining a visual push with a non-visual portion. Expects a message in the form: `This is what the user won't see;This is something he will see`. E.g. you can bundle a special ID or even a JSON string in the hidden part while including a friendly message in the visual part. When active this will trigger the `push(String)` method twice, once with the visual and once with the hidden data.
- `4` – Allows splitting a visual push request based on the format `title;body` to provide better visual representation in some OS's.
- `5` – Sends a regular push message but doesn't play a sound when the push arrives

- 100 – Applicable only to iOS. Allows setting the numeric badge on the icon to the given number. The body of the message must be a number e.g. unread count.
- 101 – identical to 100 with an added message payload separated with a space. E.g. 30 You have 30 unread messages will set the badge to "30" and present the push notification text of "You have 30 unread messages".

Push Details

E.3

When sending a push message we need some details in order to send a push message to the right device. These details provide us with the authorization required for push. Otherwise, anyone could send a push notification to any device...
Google and Apple have very different approaches to push. In the following sections I describe how to get the values you need from them. In the Uber Clone app we set these values in the Globals class constants.

Google

E.3.1

Android Push goes thru Google servers, and to do that, we need to register with Google to get keys for server usage.

We need one important value: GOOGLE_PUSH_AUTH_KEY (for the Globals class). To generate this value follow these steps:

- Login to console.cloud.google.com/
- Select APIs & Services
- Select Library
- Select Developer Tools
- Select Google Cloud Messaging
- Click Enable and follow the instructions
- The value we need is the API key, which you can see under the credentials entry

Apple

E.3.2

You will need to re-run the certificate wizard for the driver project. If you generated certificates before say no to the step that asks you to revoke them and copy your existing credentials (certificate P12 file and password) to the new project. Make sure to check the Include Push flag in the wizard so the generated provisioning includes push data.

Once this is done you should receive an email that includes the certificate details. This will include URL's for the push certificates we generated for you and the passwords for those certificates.

Apple has two push servers:

- Sandbox - use this during development
- Production - this will only work for shipping apps

You need to toggle the `APNS_PRODUCTION` flag (in the `Globals` class) when building a release version of the app.

E.4

E.4. Push Registration and Interception

Once this is out of the way we can start handling the push messages. To do that we need to implement the `PushCallback` interface in the main class of our app.

> ***This MUST be in the Main Class***
>
> The `PushCallback` interface must be defined in the main class. Otherwise it won't work correctly

A simple push listener works similarly to this code. Notice I trimmed the boilerplate so the push code stands out:

E

Listing 18. 1. DriverApp with Push

```
public class MyApp implements PushCallback {
    public void init(Object context) {
        // trimmed init code
    }
    public void start() {
        // trimmed start code
        callSerially(() -> {
            registerPush();
        });
    }
    public void stop() {
        // trimmed stop code
    }
    public void destroy() {
        // trimmed destroy code
    }
    public void push(String value) {
        Log.p("Received push callback: " + value);
    }
    public void registeredForPush(String deviceId) {
        Log.p("Registered for push device key: " +
            Push.getPushKey());
    }

    public void pushRegistrationError(String error,
            int errorCode) {
        Log.p("Error registering for push: " + error);
    }
}
```

We need to implement the PushCallback interface in the main class

registerPush should be invoked every time. Notice I use callSerially to defer the permission prompt so it appears after the Form is shown

The push callback is invoked when push is received from the server

When registration succeeds this method is invoked. Notice the deviceId isn't the push key! It's the native OS key, it's here for compatibility only

The push key is what we use to identify this device and send push messages to it. This method usually sends that key to the server so we can receive push messages here

If there was an error in registration this method is invoked

E

Appendix F: How Does Codename One Work?

F

Codename One uses a SaaS based approach so the information in this appendix might (and probably will) change in the future to accommodate improved architectures. I included this information for reference only, you don't need to understand this in order to follow the content of the book…

Since Android is already based on Java, Codename One is already native to Android and "just works" with the Android VM (ART/Dalvik).

On iOS, Codename One built and open sourced ParparVM, which is a very conservative VM. ParparVM features a concurrent (non-blocking) GC and it's written entirely in Java/C. ParparVM generates C source code matching the given Java bytecode. This effectively means that an xcode project is generated and compiled on the build servers. It's as if you handcoded a native app and is thus "future proof" for changes that Apple might introduce. E.g. Apple migrated to 64bit and later introduced bitcode support to iOS. ParparVM needed no modifications to comply with those changes.

> Codename One translates the bytecode to C which is faster than Swift/Objective-C. The port code that invokes iOS API's is hand coded in Objective-C

For Windows 10 desktop and Mobile support, Codename One uses iKVM to target UWP (Universal Windows Platform) and has open sourced the changes to the original iKVM code.

JavaScript build targets use TeaVM to do the translation statically. TeaVM provides support for threading using JavaScript by breaking the app down in a rather elaborate way. To support the complex UI Codename One uses the HTML5 Canvas API which allows absolute flexibility for building applications.

For desktop builds Codename One uses javafxpackager, since both Macs and Windows machines are available in the cloud the platform specific nature of javafxpackager is not a problem.

Lightweight Architecture

What makes Codename One stand out is the approach it takes to UI: “lightweight architecture”.

Lightweight architecture is the “not so secrete sauce” to Codename One’s portability. Essentially it means all the components/widgets in Codename One are written in Java. Thus their behavior is consistent across all platforms and they are fully customizable from the developer code as they don’t rely on OS internal semantics. This allows developers to preview the application accurately in the simulators and GUI builders.

One of the big accomplishments in Codename One is its unique ability to embed “heavyweight” widgets into place among the “lightweights”. This is crucial for apps such as Uber where the cars and widgets on top are implemented as Codename One components yet below them we have the native map component.

Codename One achieves fast performance by drawing using the native gaming API’s of most platforms e.g. OpenGL ES on iOS. The core technologies behind Codename One are all open source including most of the stuff developed by Codename One itself, e.g. ParparVM but also the full library, platform ports, designer tool, device skins etc.