CREATE AN UBER CLONE IN 7 DAYS

BUILD A REAL WORLD FULL STACK MOBILE APP IN JAVA

SHAI ALMOG

For online information and ordering of this and other Manning books, please visit www.codenameone.com.

Create an Uber Clone in 7 Days

BUILD A REAL WORLD FULL STACK MOBILE APP IN JAVA

Shai Almog
Codename One Academy

Contents

* Preface

Audience 1
Prerequisites 2
What you don't need to Know 2
Get this Book for FREE! 2
How to Read this Book? 3
Software Prerequisites 4
Getting Help 4
Using the Code 4
Important Notice About Cloning and Copyrights 5
Thanks and Acknowledgments 5

1 Hello World

1.1. What's Codename One? 7
1.1.1. Build Cloud 8
1.2. Getting Started 8
1.2.1. New Project 9
Running 11
The Source Code 12
1.2.2. Run on Device 18
Signing 18
Build and Install 25
1.3. How Does it Work? 26
1.3.1. Mobile is Different 27
Density (DPI/PPI) 27
1.3.2. Performance 30
App Size 30
Power Drain 31
1.3.3. Sandbox and Permissions 31
1.4. Summary 32

2 Core Concepts

2.1. The Todo App 35
2.2. Layouts and Hierarchy 37
2.2.1. Layout Managers 46
Terse Syntax 49
2.2.2. Styles 49
Designer Tool 52
2.2.3. Event Handling 70
Observers and Event Types 71
Event Dispatch Thread 73
2.2.4. IO and Storage 74
2.3. Summary 76

3 Spring Boot Overview

3.1. What's Spring Boot? 77
3.2. Why Spring Boot 78
3.3. What will we Use? 78
3.4. Summary 79

4 Day 1: The Mockup

4.1. Deconstructing Uber 81
4.1.1. Portrait Only UI 83
4.1.2. iOS and Android Look Almost Identical 83
4.1.3. Inconsistent Titles and Simple Design 84
4.2. The UI Elements 84
4.2.1. Setup 84
Common Styles 86
4.2.2. Countries Button 89
4.2.3. CountryPickerForm 92
initBlackTitleForm 95
4.2.4. Login 98
Shadow 104
4.2.5. Social Login 106
4.2.6. Enter Mobile Number 107
4.2.7. SMS Verification 110
4.2.8. Enter Password 116
4.3. Summary 118

5 Day 1: The Mockup – Night Hack

5.1. The Map 119
5.1.1. Map Layout 120
5.1.2. The MapForm 127
5.1.3. Navigation UI 133
Navigation UI - Destination 141
5.1.4. The Side Menu 147
5.2. Summary 150

6 Day 2: The Server

6.1. Server Features 153
6.2. Storage Schema 155
6.2.1. The User Object 155
JPA 159
6.2.2. Ride and Waypoint 164
6.2.3. Rating 167
6.2.4. CRUD Interfaces 168
6.2.5. Data Access Objects (DAO) 170
6.3. UserService 172
6.3.1. SecurityConfiguration 177
6.4. UserWebservice 178
6.5. Location Updates Through WebSockets 183
6.5.1. WebSocketConfig 184
6.5.2. LocationService 186
6.5.3. WebSocket Handler 187
6.6. Summary 191

7 Day 3: Connecting the Client/Server and SMS Activation

7.1. Before we Begin 193
7.2. User and Properties 196
7.2.1. UserService 199
7.2.2. SMS Activation Process 204
7.3. Location and WebSockets 211
7.3.1. Updating the Map 221
7.4. Lets See it Working 224
7.5. Summary 224

8 Day 4: Search Route and Hailing

8.1. SearchService 228
8.1.1. Reverse Geocoding 229
8.1.2. Places Autocomplete 232
8.1.3. Directions 239
decodePolyline 240
DirectionResults 242
The Directions Method 242
8.2. Search 243
8.2.1. AutoCompleteAddressInput 244
8.2.2. CompletionContainer 251
8.2.3. MapForm 256
8.3. Summary 265

9 Day 4: Search Route and Hailing – Night Hack

9.1. Route 267
9.1.1. Tags 268
BlackAndWhiteBorder 272
9.1.2. Navigation Mode 277
9.2. Hail a Car 283
9.2.1. BlinkDot 285
9.2.2. LocationService 287
9.3. Summary 295

10 Day 5: Driver App and Push

10.1. Server 297
10.1.1. User 297
10.1.2. RideDAO 298
10.1.3. Ride, RideRepository and Waypoint 299
Ride 300
RideRepository 301
Waypoint 301
10.1.4. RideService 302
10.1.5. RideWebservice 305
10.1.6. UserService, LocationService and UserWebservice 307
10.1.7. Handler 310
10.2. The Driver App 315
10.2.1. How does the Appstore Identify the App? 315
10.2.2. Customizing the Driver App 317
10.2.3. Push Notification 319
Before we Begin with Push 319
Push Registration and Interception 319
10.2.4. DriverService and Ride 321
10.2.5. SearchService Changes 323
10.2.6. UserService Changes 324
10.2.7. MapForm Changes 327
10.3. Summary 339

11 Day 6: Billing and Social Activation

11.1. Billing 341
11.1.1. Braintree 342
11.1.2. Server Side 343
BraintreeService 344
The nonce 346
The Actual Payment 347
11.1.3. Client Side 349
11.2. Social Login 351
11.2.1. Client Side 353
11.2.2. Driver and User Apps 354
11.2.3. Facebook Login 354
iOS Wizard 354
Android Wizard 355
integration 356
11.2.4. Full Integration 358
11.2.5. Google Login 362
11.3. Summary 364

12 Day 7: Transitions and Refinement

12. Day 7: Transitions and Refinement 365
12.1. Login Form Rotation Animation 365
12.2. Cover and Material Transition 370
12.2.1. Morph Transitions 370
12.2.2. Cover Transition 373
12.3. Circle Animated FloatingActionButton 375
12.4. User Information 379
12.4.1. Settings Form 379
12.4.2. EditAccountForm 385
Avatar 390
12.5. Summary 391

13 Summary and Moving Forward

13.1. Congratulations 393
13.2. Key Takeaways and Lessons Learned 394
13.3. Different Approaches and Next Steps 395
13.3.1. TDD (Test Driven Development) 395
13.3.2. GUI Builder 395
13.3.3. CSS 396
13.3.4. WebSockets Everywhere 396
13.3.5. Kotlin 396
13.4. Summary 396

A Appendix A: Setup Codename One

A.1. IntelliJ/IDEA 397
A.2. NetBeans 398
A.3. Eclipse 399

B Appendix B: Setup Spring Boot and MySQL

B.1. MySQL Setup 401
B.2. Setup Spring Boot Project 402

C Appendix C: Styling Codename One Apps with CSS 407

C.1. Getting Started with CSS 408
C.1.1. Selectors 409
C.1.2. Properties 410
C.1.3. Images 411
C.2. Summary 412

D Appendix D: Installing cn1libs 413

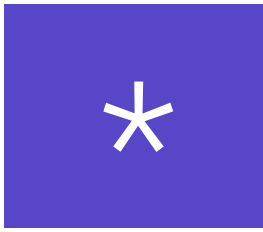
E Appendix E: Push Notification

E.1. What Is Push? 417
E.2. Various Types of Push Messages 418
E.3. Push Details 419
E.3.1. Google 419
E.3.2. Apple 419
E.4. Push Registration and Interception 420

F Appendix F: How Dows Codename One Work? 423

F.1. Lightweight Architecture 424

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 facebook 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 IP!

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 which supported the process of the books development throughout. 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 reviews were anonymized I don't have access to it. Thank you for taking the time to read the earlier rough drafts and provide valuable feedback that undoubtedly improved this book immensely.

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

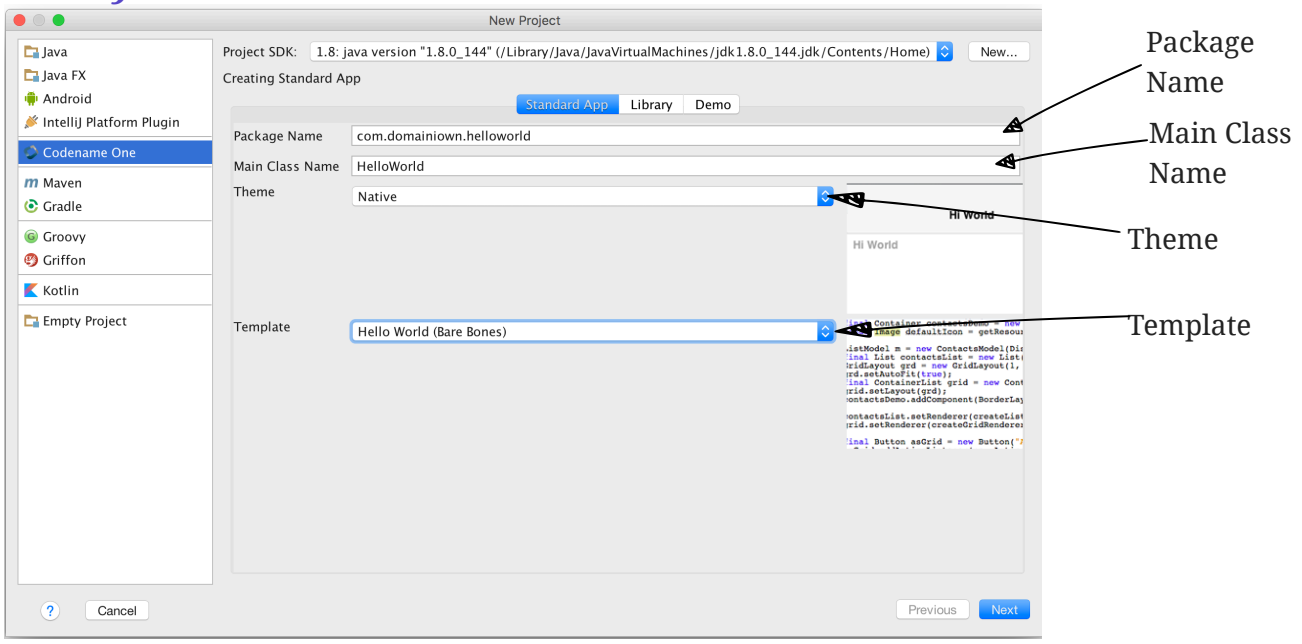
Before we get to the code there are few important things we need to go over with the new project wizard.

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:

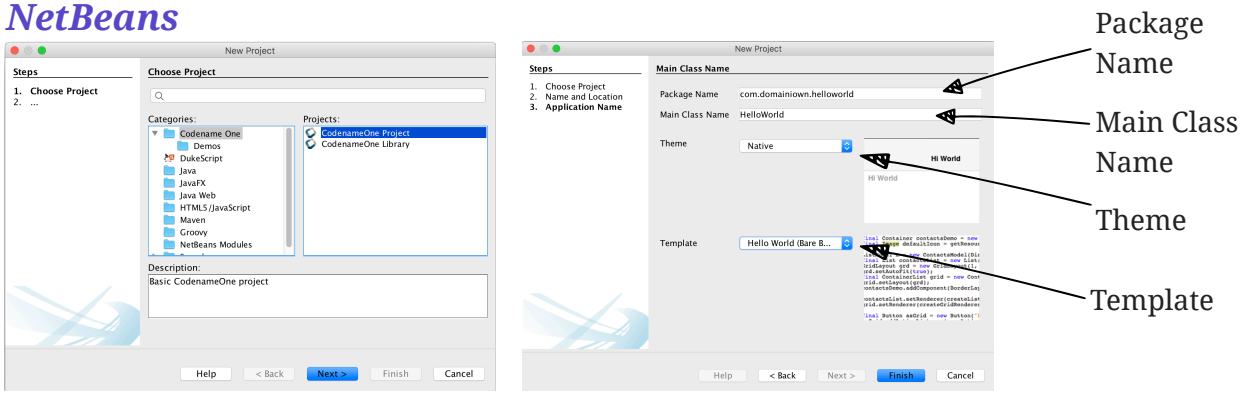
- **App Name** - This is the name of the app and the main class, it's important to get this right as it's hard to change this value later
- **Package Name** - It's **crucial** you get this value right. Besides the difficulty of changing this after the fact, once an app is submitted to iTunes/Google Play 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

1

IntelliJ



NetBeans



Eclipse

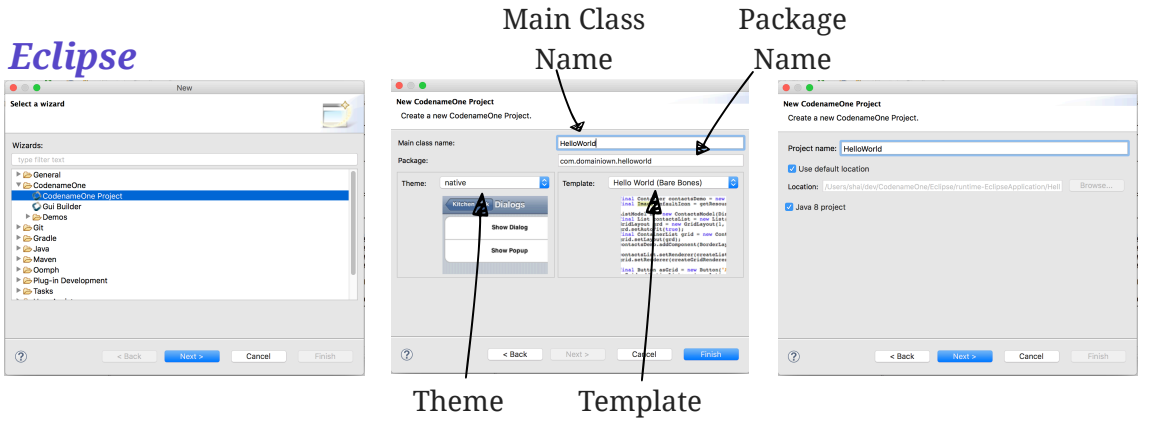


Figure 1. 1. The New App Wizard

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

1 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.

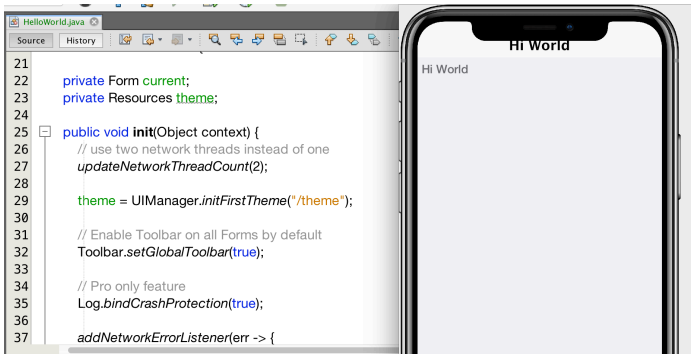


Figure 1. 2. HelloWorld Running on the Simulator with an iPhone X Skin

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

1 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()

```
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();    ⑤
}
```

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

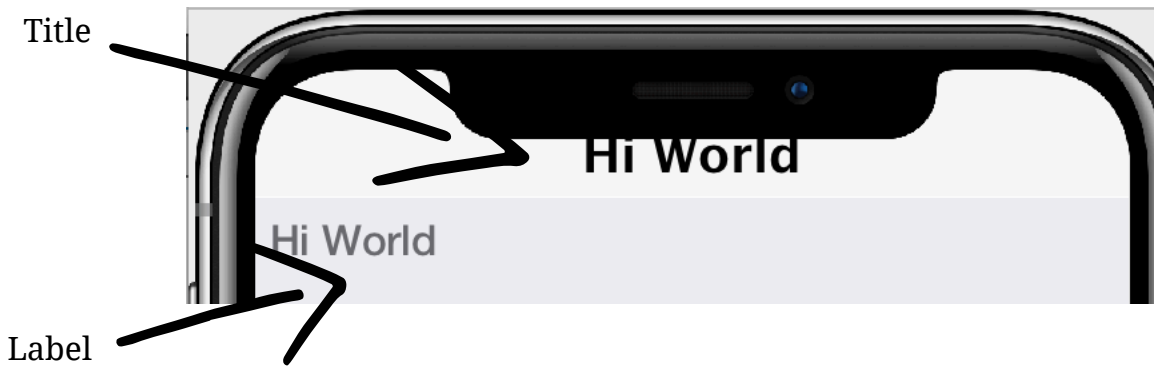


Figure 1. 3. Title and Label in the UI

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).

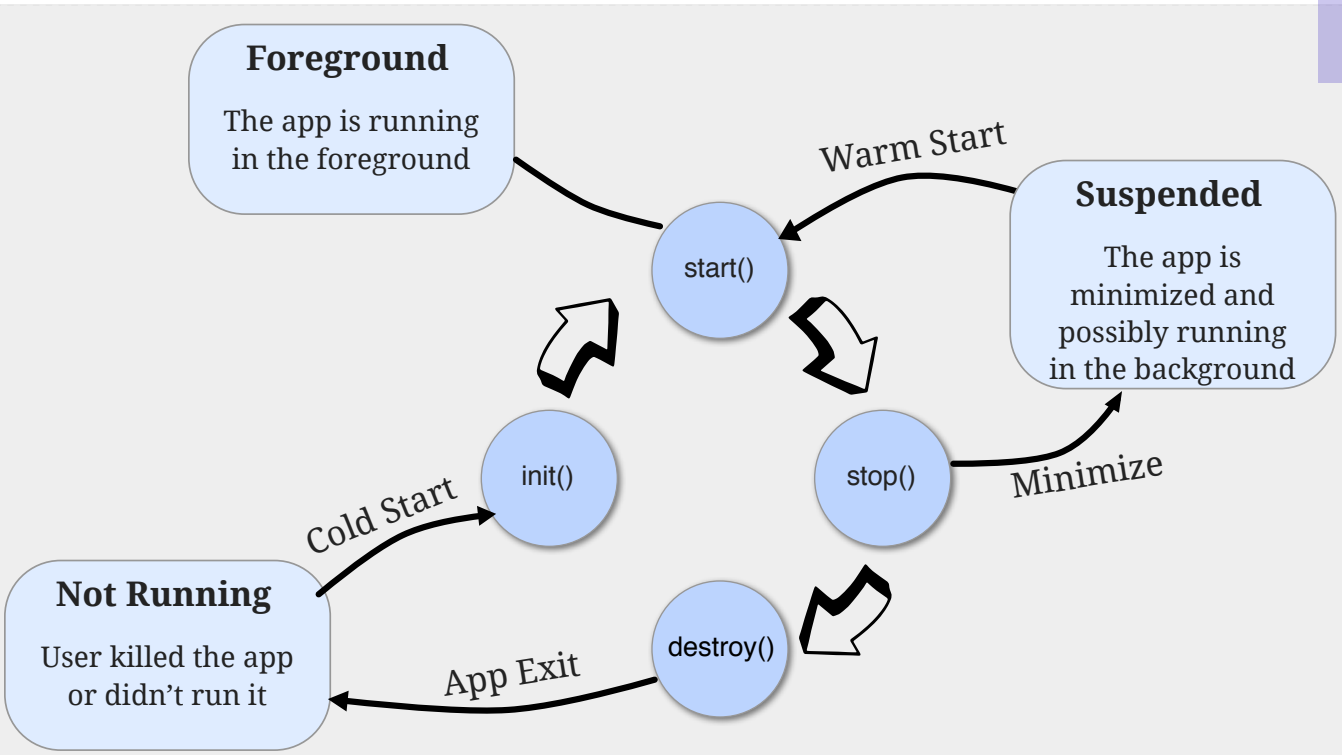


Figure 1. 4. Codename One Application Lifecycle

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();
    }
}

public void destroy() {
}
```

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

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.

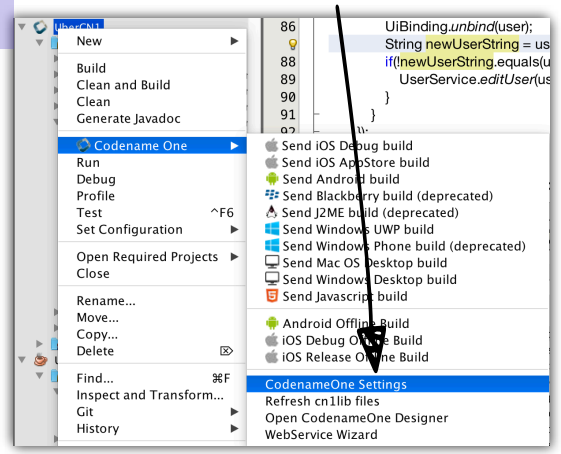
With that in mind generating an Android certificate is trivial.

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

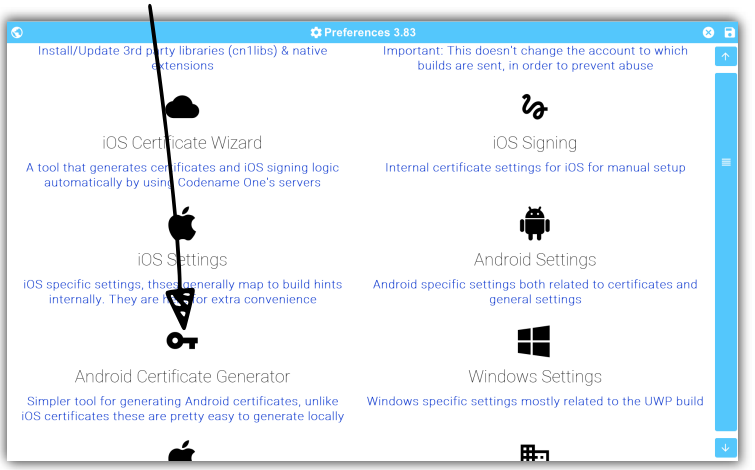
Your certificate will generate into the file `Keychain.ks` ***in your home directory***

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”



Click “Android Cetificate Generator”



Don’t forget the password

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

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 lets 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.

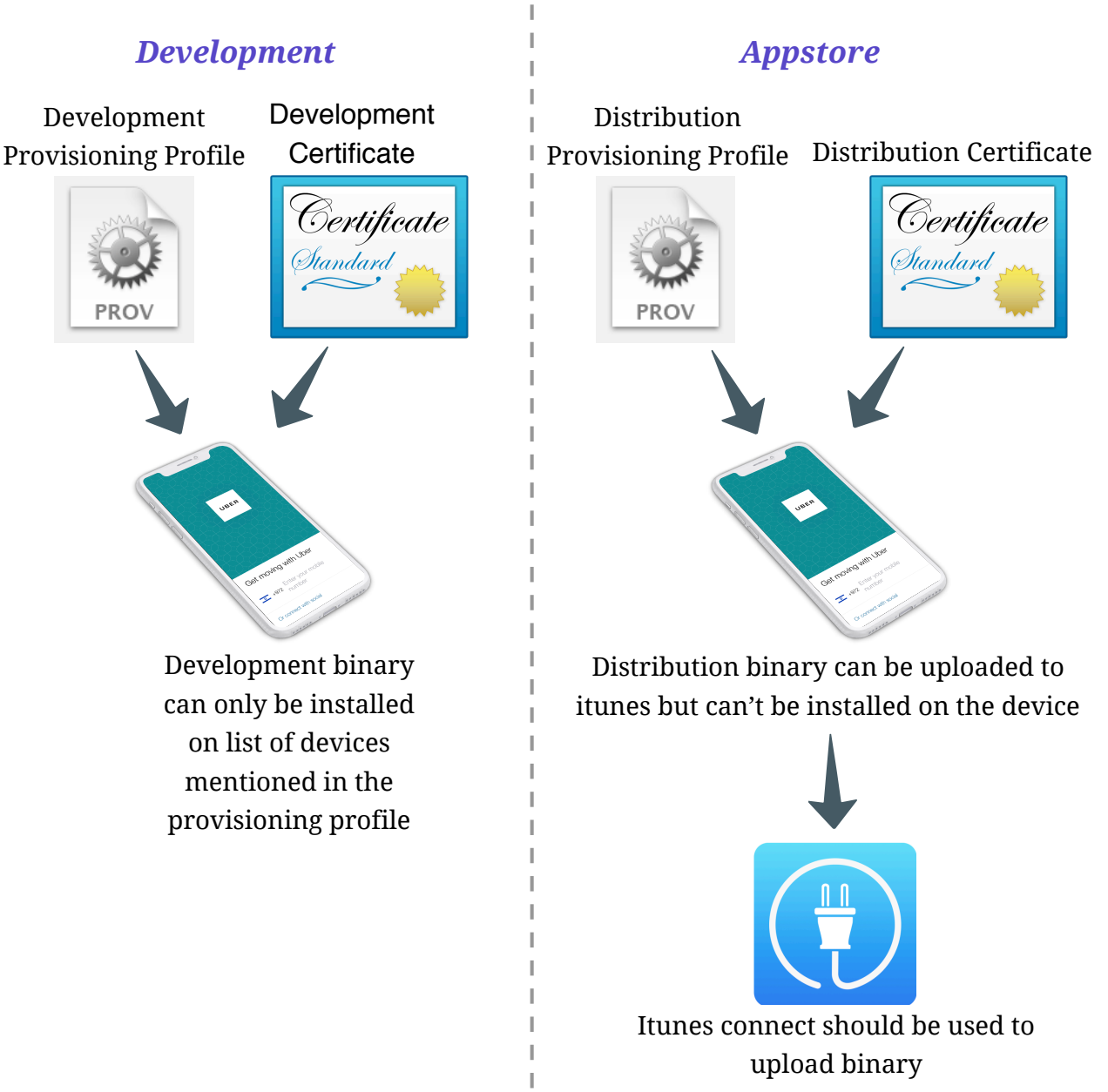


Figure 1. 6. The Four Files Required for iOS Signing and Provisioning

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

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

Next to proceed

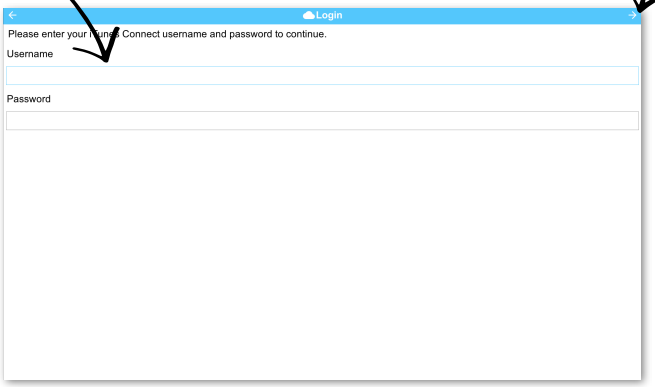
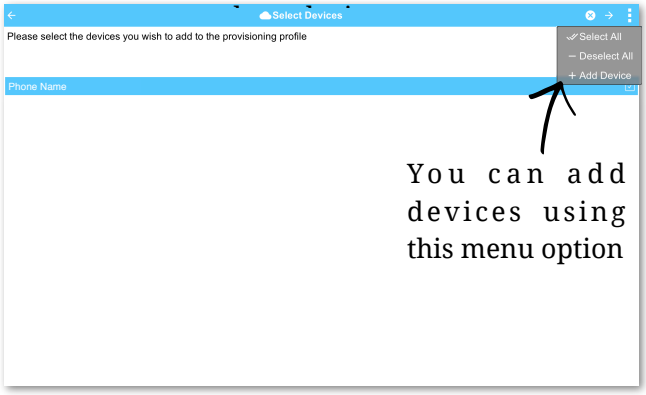
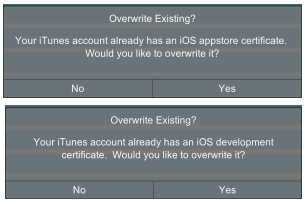


Figure 1. 7. Using the iOS Certificate Wizard Steps 1 and 2

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



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



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

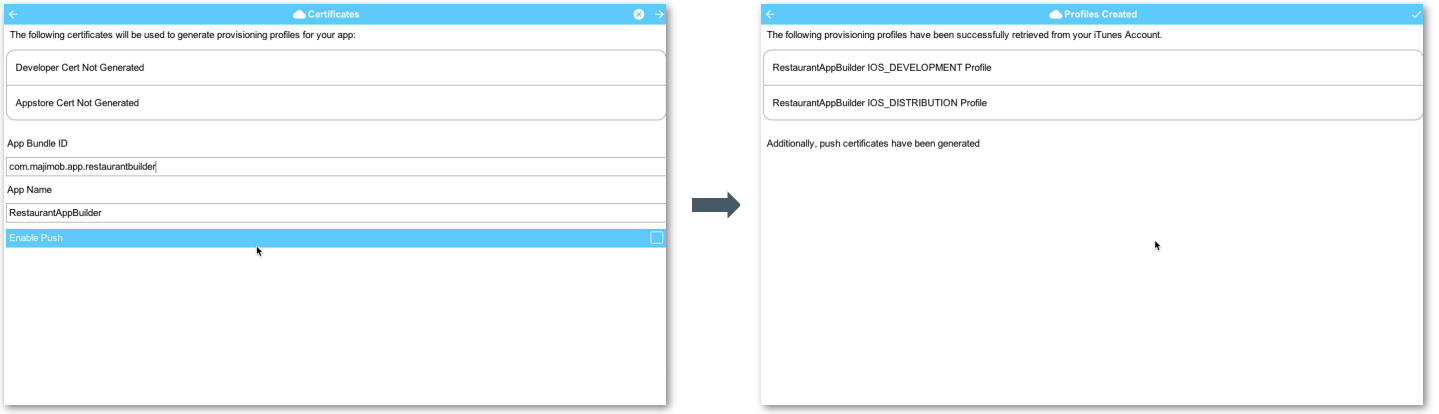


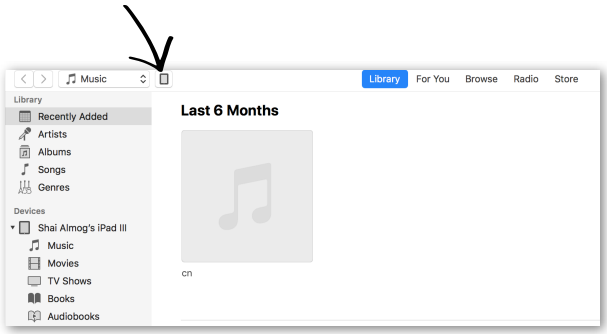
Figure 1. 9. Using the iOS Certificate Wizard Steps 5 and 6

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 Apples iOS developer website.

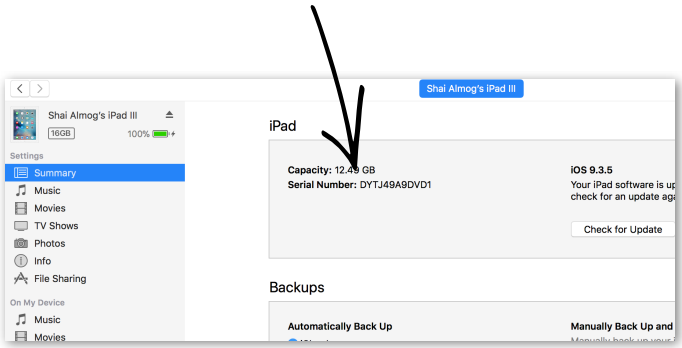
One important aspect of provisioning on iOS is the device list in the provisioning step. Apple only allows you to install the app on 100 devices during development. This blocks developers from skipping the appstore altogether. It's important you list the correct UDID for the device in the list otherwise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



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

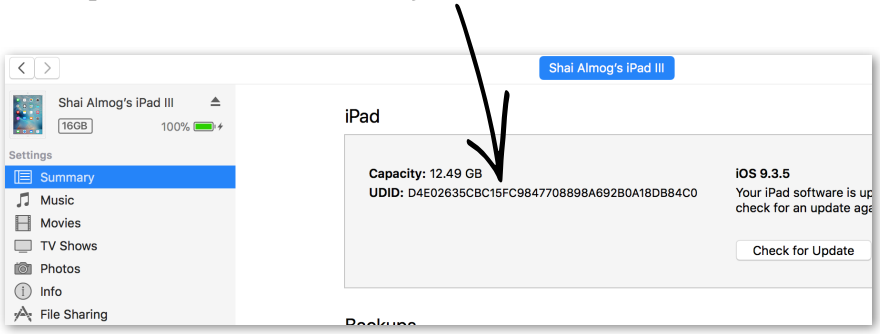


Figure 1. 10. Get the UDID of a Device

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.

Adhoc 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 signup 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.

1 

The first time you send a build you would 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:

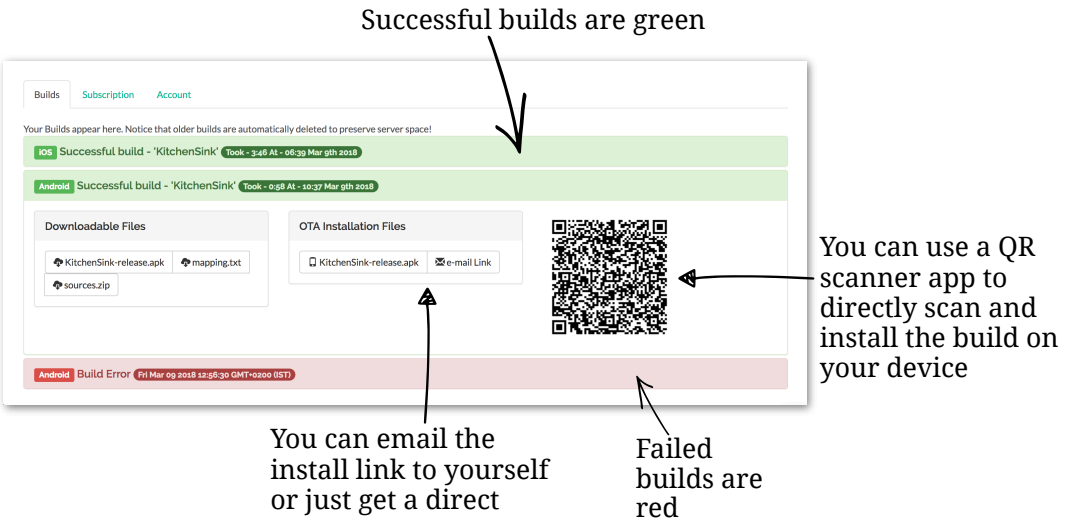


Figure 1. 11. Build Results

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 once 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.

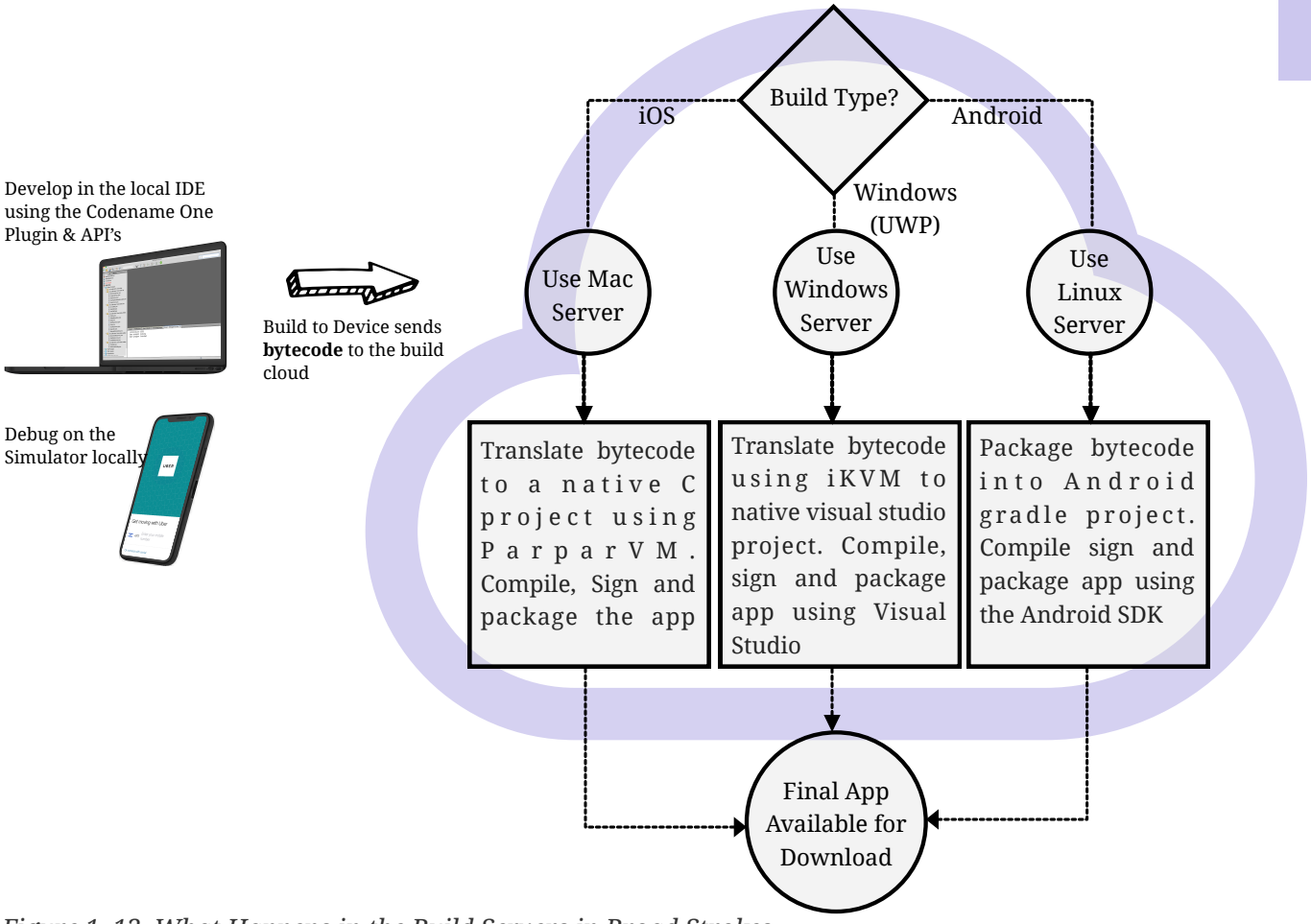


Figure 1. 12. What Happens in the Build Servers in Broad Strokes

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 1.3.1

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!

1 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!

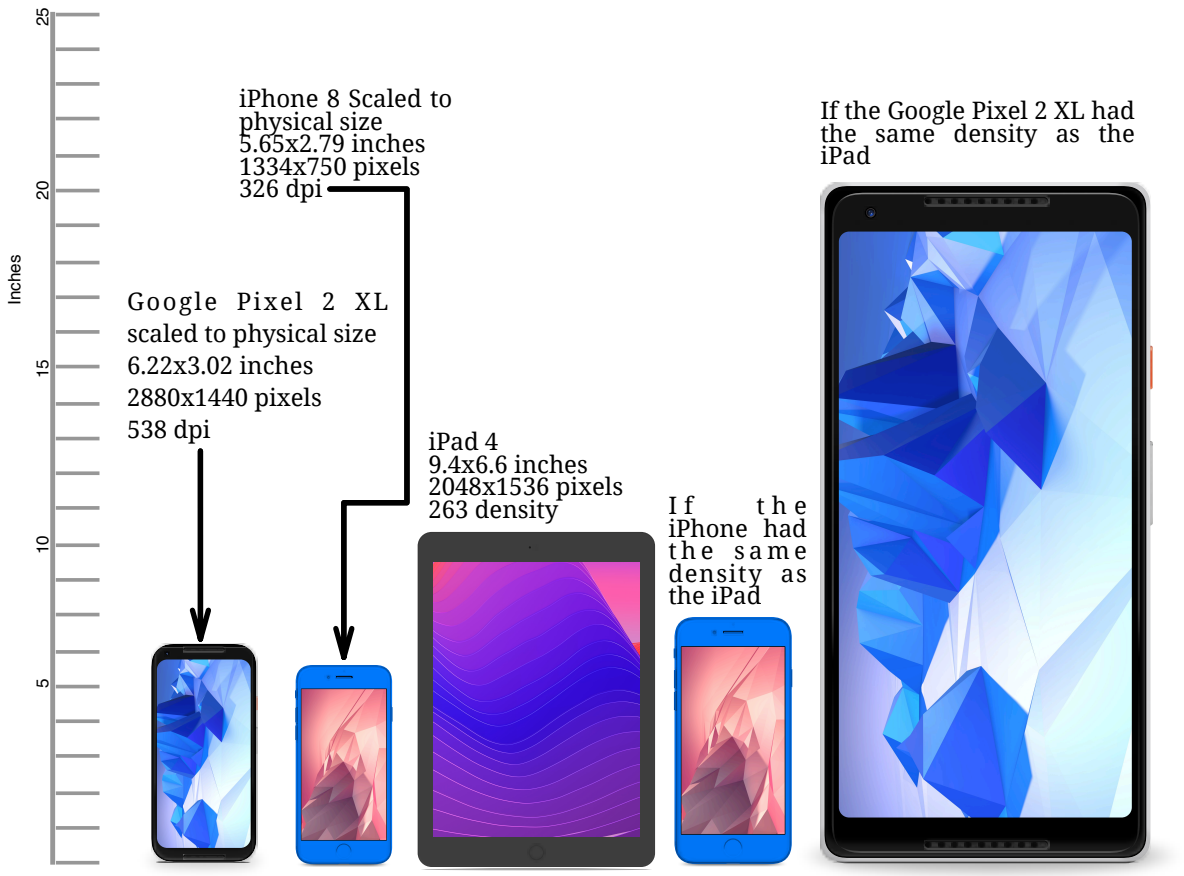


Figure 1. 13. Device Density vs. Resolution

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.

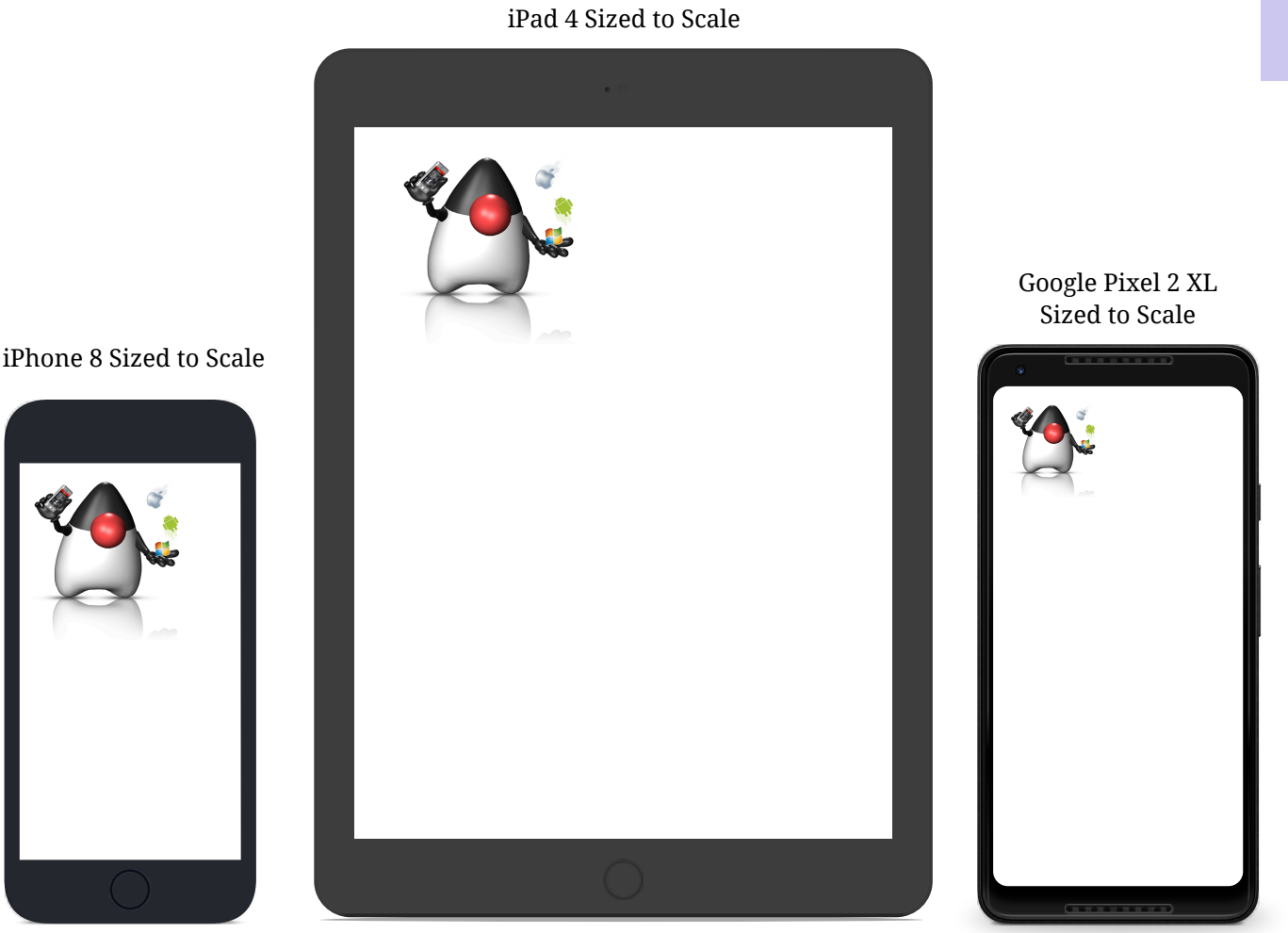


Figure 1. 14. How the Same Image Looks in Different Devices

The exact same image will look different on all of the devices 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 later in the chapter.

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 1.3.3

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.

1 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 decades 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/