# Java Plays – Mobile Starter Kit - Starter (Basic)

**Presented by:**

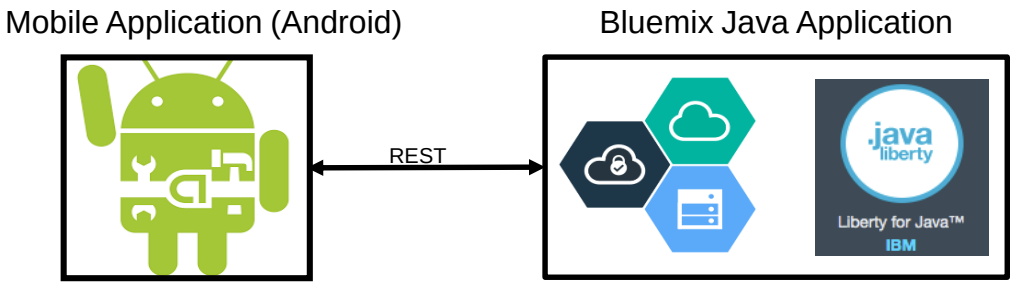
IBM

IBM **Cloud**

# Mobile Starter Kit

## Starter

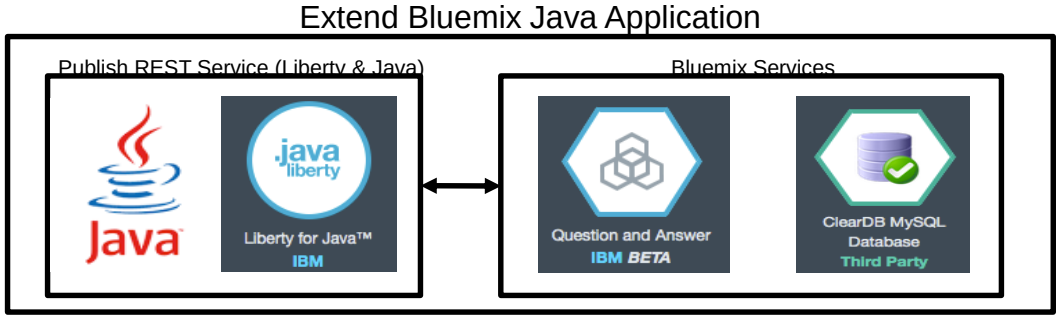
- Basic Mobile Application (Android)
- Basic Bluemix Java Application – REST Services




USE CASE – A Cognitive health care solution that provides information about a disease or a symptom

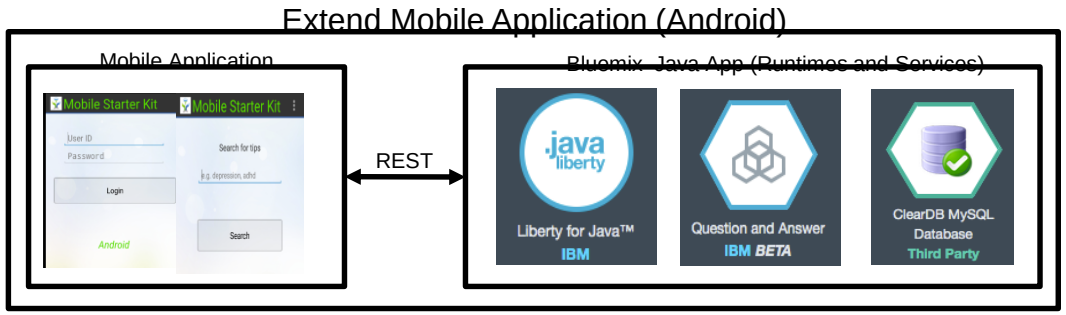
## Extend Bluemix Java App

- Extend the Basic RESTful Service to perform real time operations




## Extend Mobile Application

- Extend the Mobile App to Integrate with the REST Services (Login, Searchtips)




© 2015 IBM Corporation

# Agenda

- Pre-Requisites

- Bluemix Java Application
  - Configure in Eclipse
    - Setup Bluemix
    - Import the Code from Git/Jazzhub
  - Create REST Services
    - Create Service
    - Code Walkthrough
  - Build & Deploy
    - Build the Application
    - Deploy to Bluemix
  - Test

- Mobile UI (Application)
  - Configure
    - Setup AVD Manager
    - Setup SDK Manager
    - Import the Code
  - Create Application UI
    - Screen layouts/Form Inputs
    - Integrate with REST Service
    - Code Walkthrough
  - Build & Run
    - Build the Mobile Application
    - Run and Test the Application
  - Deploy and Test on Mobile

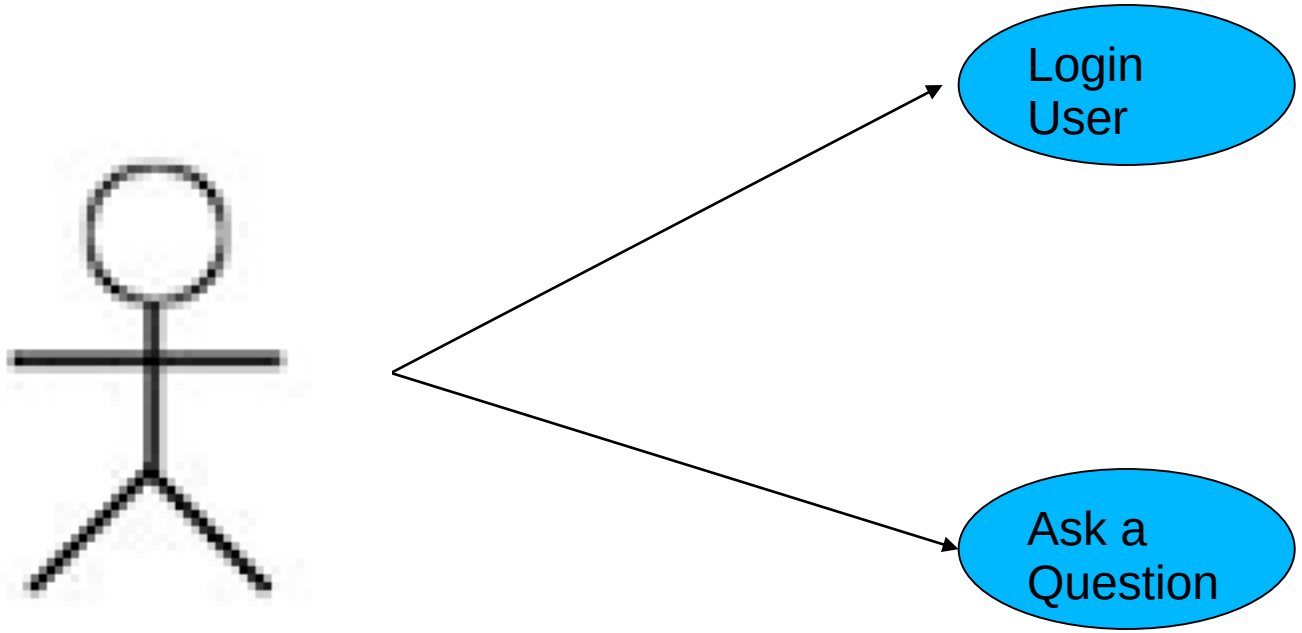
© 2015 IBM Corporation

## Pre-Requisites

- Create Bluemix Account
  - https://bluemix.net
  - https://hub.jazz.net
- JDK 1.7
- Eclipse Mars (Download – OS Specific)
  - https://eclipse.org/downloads/
- Eclipse Marketplace
  - Install Bluemix Plugin
- MySQL Workbench (Download - OS-Specific)
  - https://dev.mysql.com/downloads/workbench/

© 2015 IBM Corporation

# Bluemix Java Application – Use Case

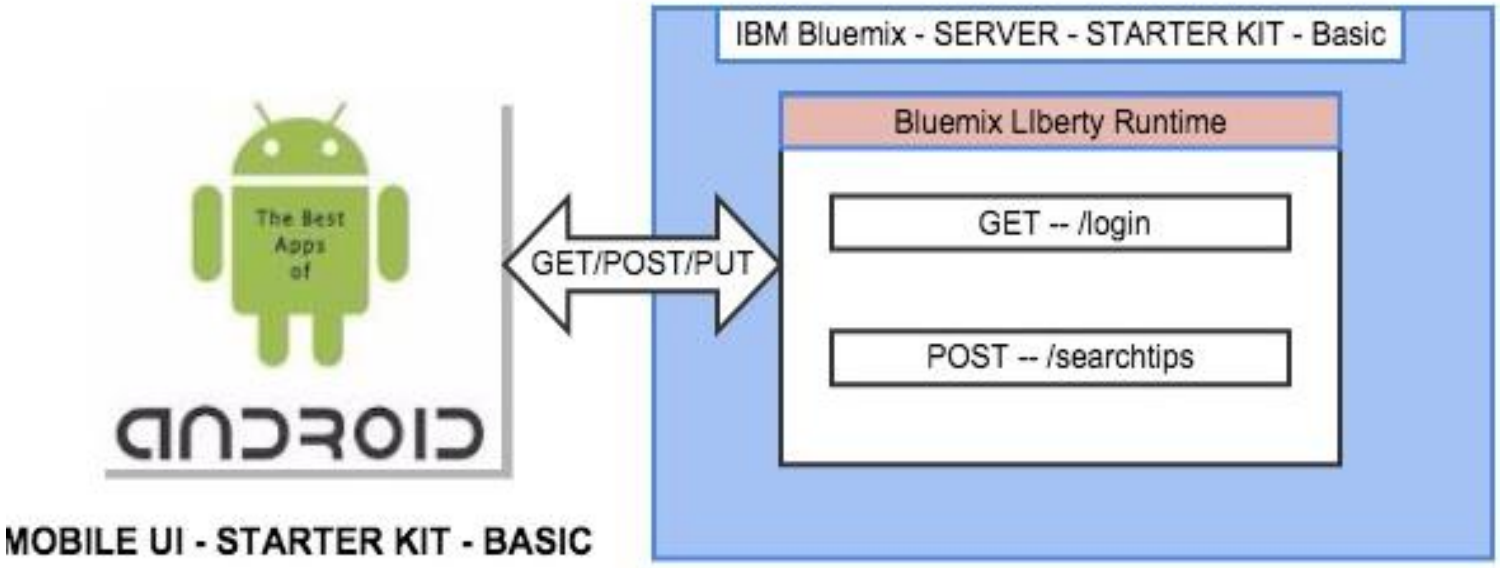



Use Case:

User logs in through a mobile app and asks a question

© 2015 IBM Corporation

# Bluemix Java Application – Basic




Basic Application:

1) Create and Expose a Simple login REST Service, Searchtips Service in Bluemix
2) Create Simple UI Screens (Login, Search) and call REST Service in Bluemix

© 2015 IBM Corporation

# Bluemix Java Application - Configuration

There are two ways download the code and pre-requisites from for the bootcam

1. Option1: Get the Pre-Installed code and repository from the instructor

   OR

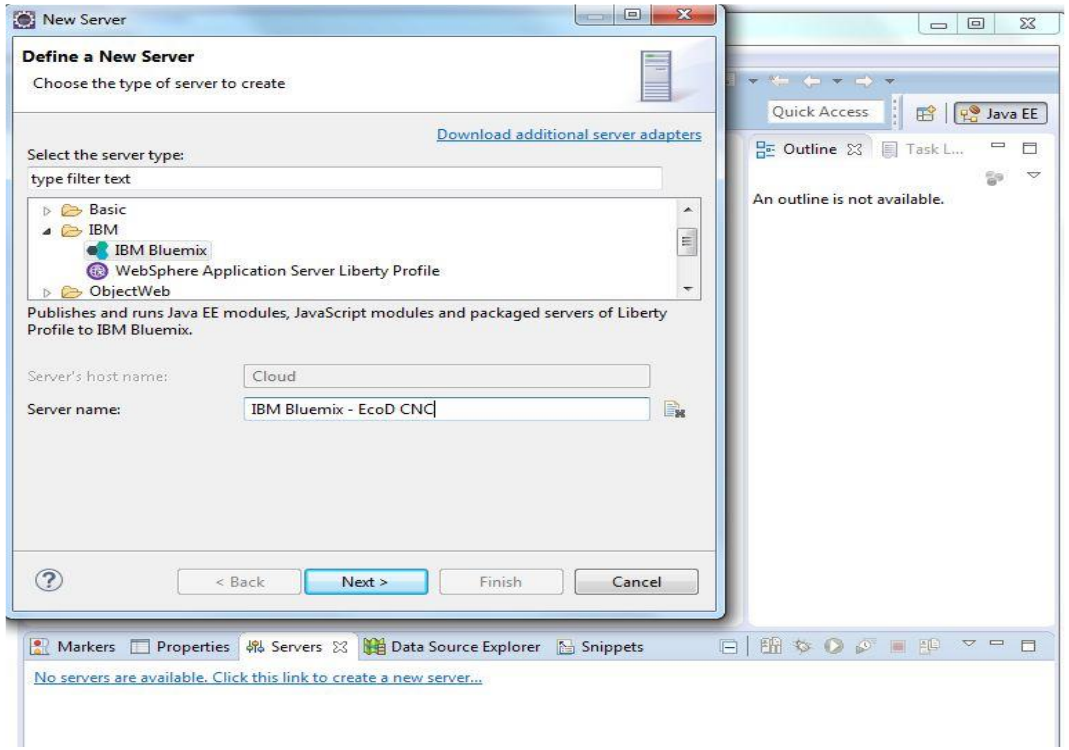
2. Option 2: Go to Jazzhub and Get it

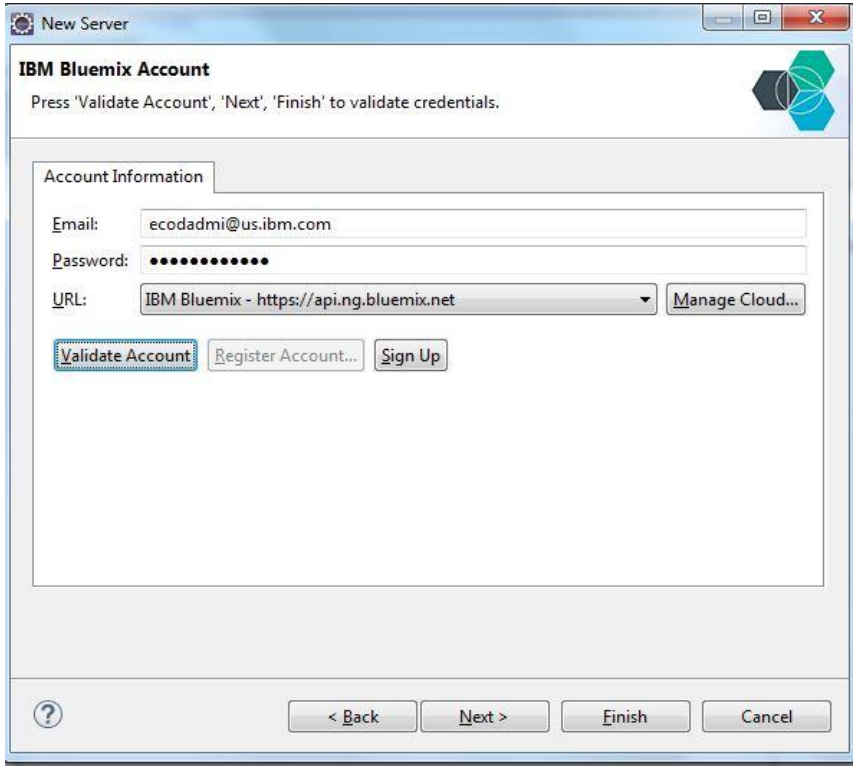
   - https://hub.jazz.net/git/ecosysdevcnc/javaplays-mobilestarterkit/
     - For Template
       - Javaplays-restjava-mobilestarterkit-template
     - For Extend – Fully Functional
       - Javaplays-restjava-mobilestarterkit-extend
     - For Basic Template Fully Functional
       - Javaplays-restjava-mobilestarterkit-basic

© 2015 IBM Corporation

# Bluemix Java Application – Configure Option2

- Navigate to JEE  perspective in eclipse (located on the left hand corner) 
- Click on the link in the bottom panel to
- Add a new Server  Select IBM Bluemix



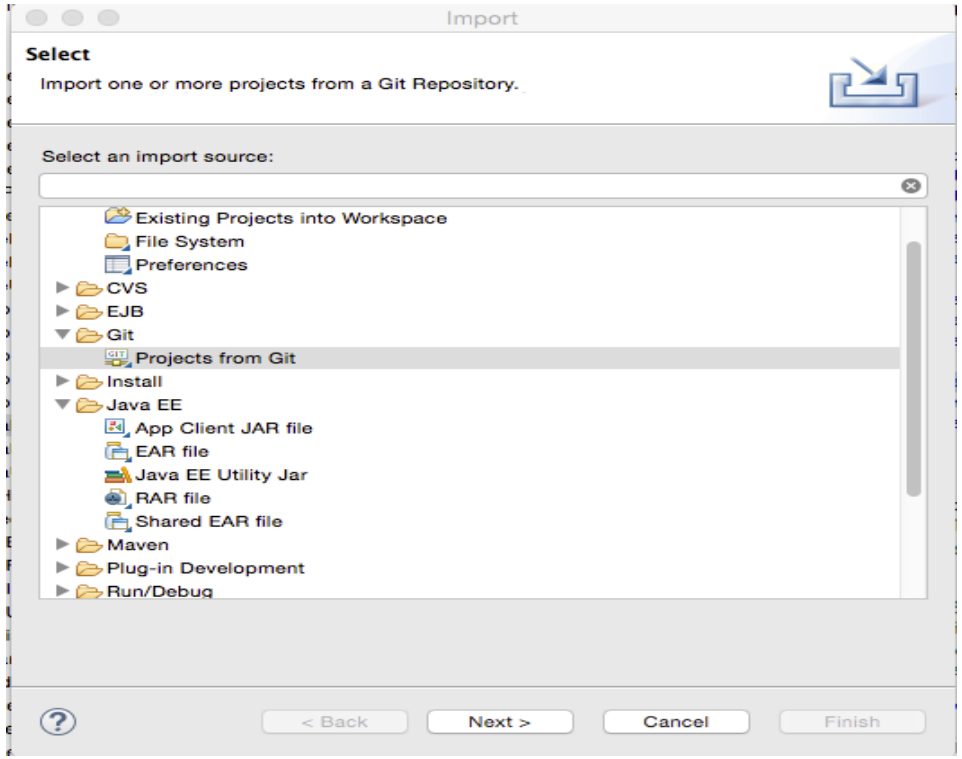



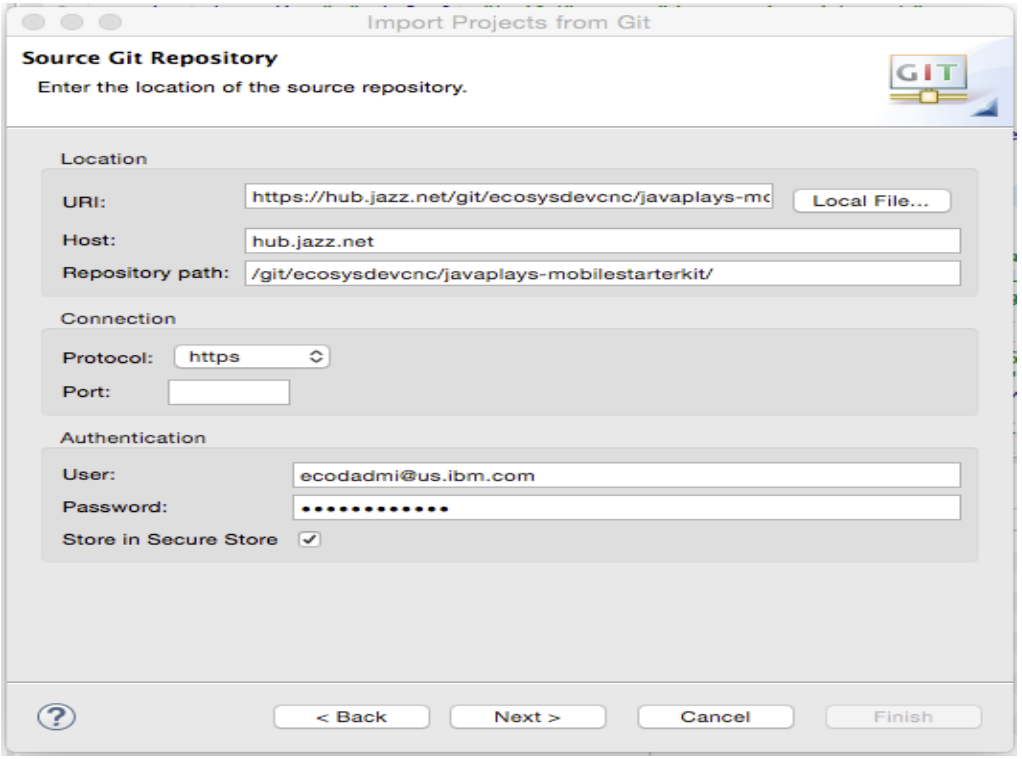

© 2015 IBM Corporation

# Bluemix Java Application – Configure Option 2- Pull the Code from JazzHub

- Start Eclipse, open a workspace, and close the Eclipse Welcome page.
- Select File → Import, Git → Projects from Git, Clone URI and clone the following URI

https://hub.jazz.net/git/ecosysdevcnc/javaplays-mobilestarterkit/

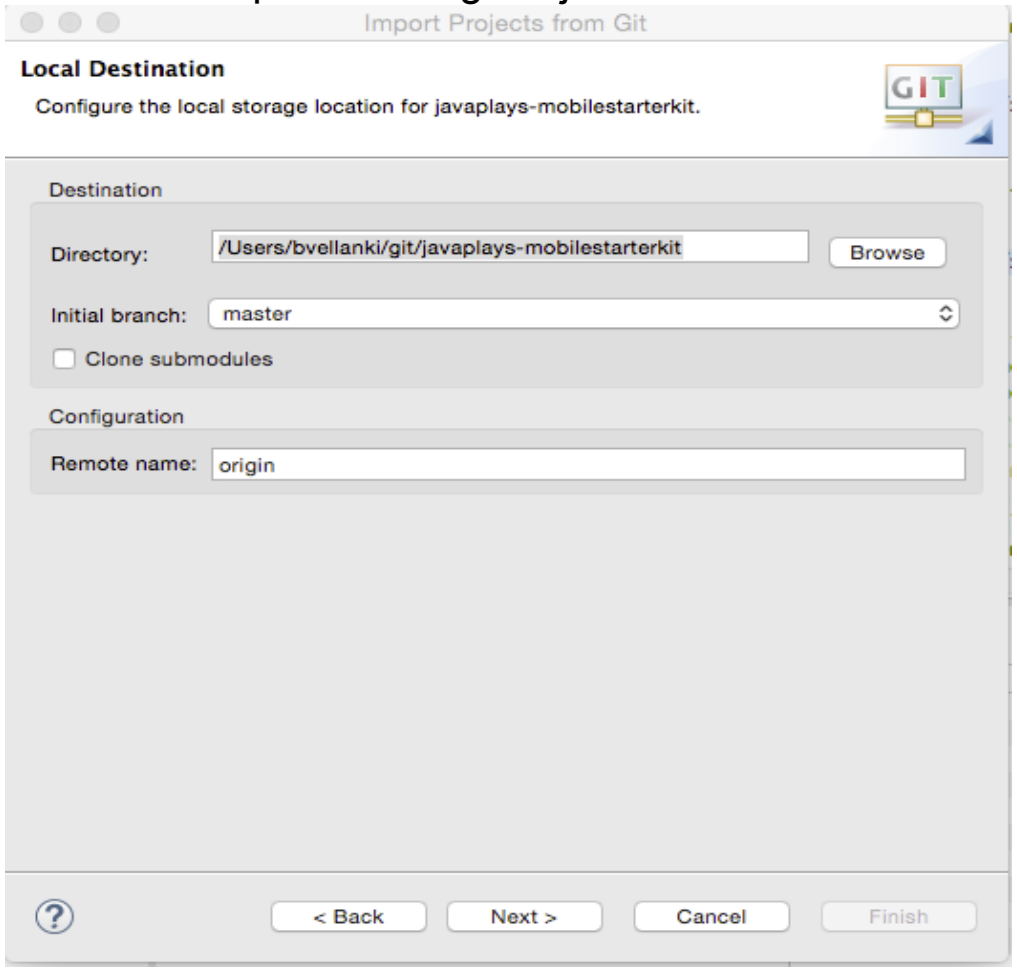


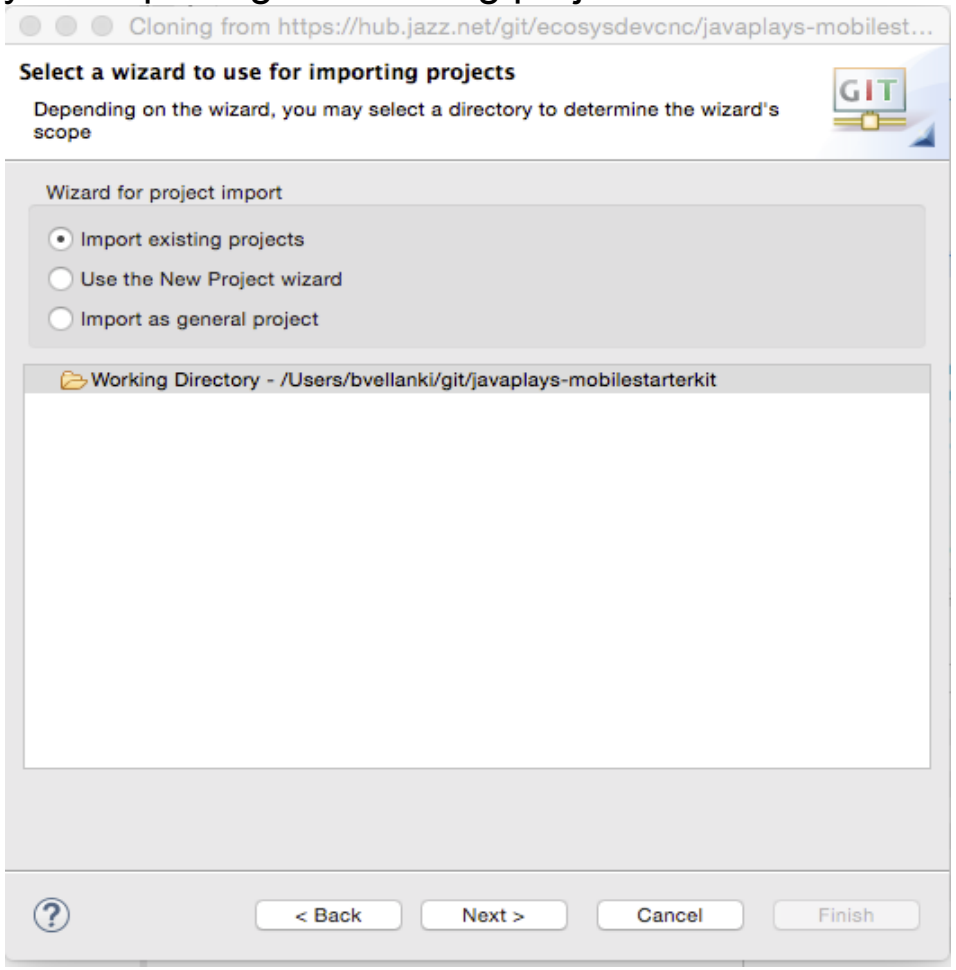




© 2015 IBM Corporation

# Bluemix Java Application – Configure Option 2 - Pull the Code from JazzHub (Cont'd)

- Import the master branch to your local destination or location (directory)
- Select "Import existing Projects" and select the directory for importing the existing project

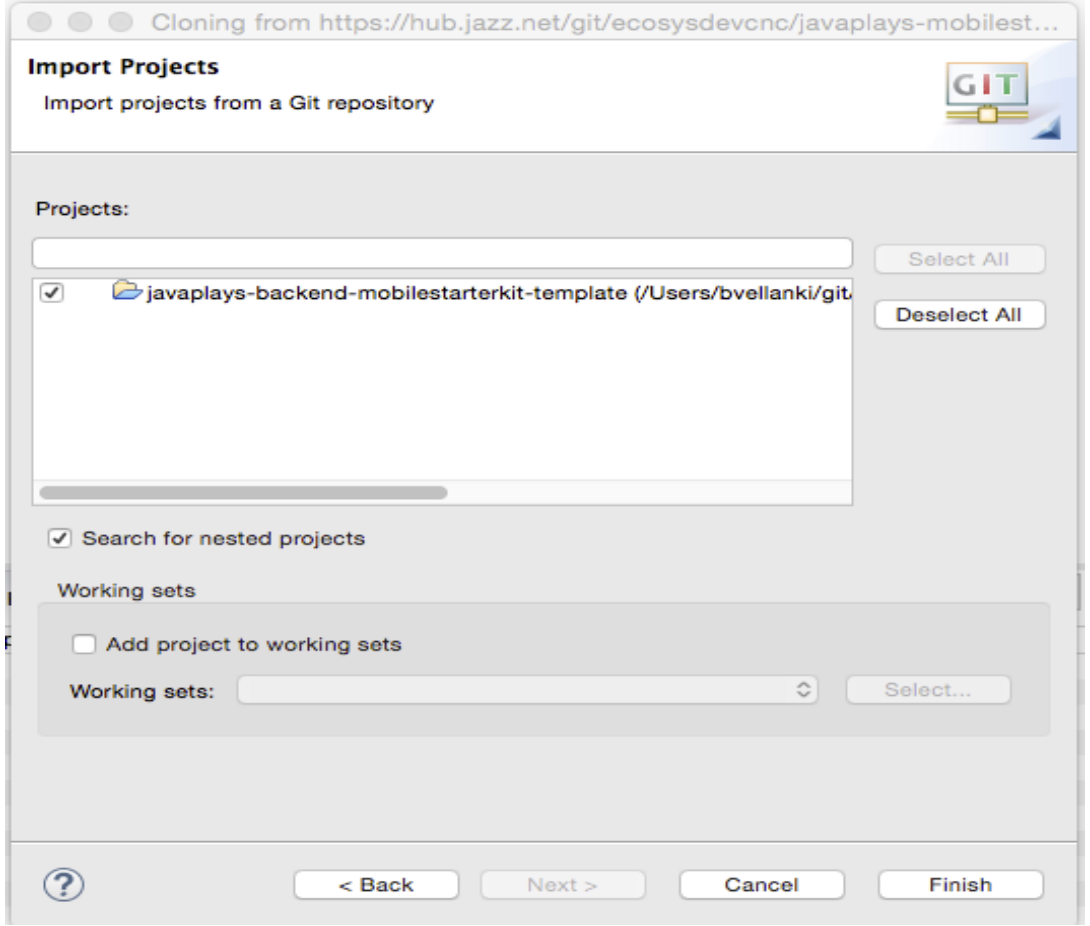



© 2015 IBM Corporation

# Bluemix Java Application – Configure Option2 - Pull the Code from JazzHub (Cont'd)

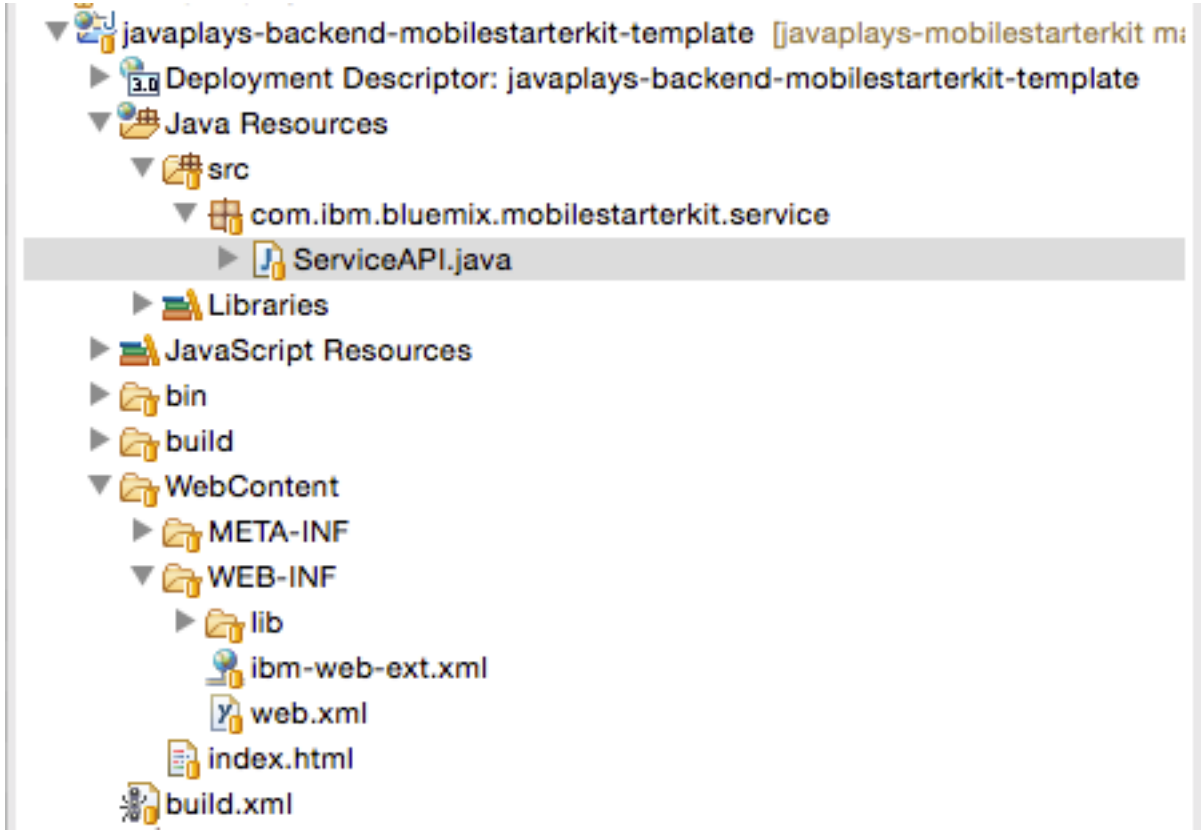
- Select the server side code to import in to your eclipse workspace for development.




© 2015 IBM Corporation

# Bluemix Java Application – Create – ServiceAPI.java

- Change the Eclipse Perspective (located on the right hand corner of the IDE) to Java EE. The server side code imported should be visible for deployment. Open ServiceAPI.java




© 2015 IBM Corporation

# Bluemix Java Application – Create

- Copy the attached code from template code (javaplays-restjava-mobilestarterkit-template/src) directory in to ServiceAPI.java

Microsoft Word Document

- Open web.xml (located under javaplays-restjava-mobilestarterkit-template/WebContent/WEB_INF).
- Copy the attached xml code or from template code directory in to web.xml

Microsoft Word Document

© 2015 IBM Corporation

# Bluemix Java Application – Create – web.xml Code Walkthrough

- Web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="3.0" xmlns="http://java.sun.com/
    <display-name>javaplays-mobilestarterkit-lab1</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
     <servlet>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <url-pattern>/api/*</url-pattern>
  </servlet-mapping>
</web-app>
```

The web.xml file contains information about the structure and external dependencies of web components in the module and describes how the components are used at run time. To enable the web container to run Java API for RESTful Web Services (JAX-RS) applications, you can configure the web.xml file to point directly to the JAX-RS servlet. When using servlets, you can define a servlet path in the web.xml file that is appended to the base URL.

Then all the @Path and @Provider classes found in your web application will be available in the "default" JAX-RS application with a servlet URL pattern of "/api/*".

© 2015 IBM Corporation

# Bluemix Java Application – Create – ServiceAPI.java Code Walkthrough

- /login POST RESTful Service

```
@Path("/service")
public class ServiceAPI {

    @Path("/login")
    @POST
    public String checkLogin(String creds) {
        try {
            JSONObject credentials = new JSONObject(creds);
            String userID = credentials.getString("user_id");
            String password = credentials.getString("password");
            if (userID.equals("admin") && password.equals("password")) {
                return "Successful";
            } else {
                return "Failed";
            }

        } catch (JSONException e) {

            e.getStackTrace();
            return "Failed";

        }
    }
```

- The all the @Path found in your Bluemix Java Application application will be available in the "default" JAX-RS application with a servlet URL pattern of "/api/*".
  - Based on the code for login as shown, the REST service can be invoked by calling the URL  http://<host:port>/api/service/login
- The /login URI POST invocation will trigger checkLogin method. This method does the following actions:
  - Receives the data passed in as creds
  - Parses the user_id and password values Validates the credentials passed
  - Returns a Response

© 2015 IBM Corporation

# Bluemix Java Application – Create – ServiceAPI.java (/login REST call)

- /login POST RESTful Service

```
@Path("/service")
public class ServiceAPI {

    @Path("/login")
    @POST
    public String checkLogin(String creds) {
        try {
            JSONObject credentials = new JSONObject(creds);
            String userID = credentials.getString("user_id");
            String password = credentials.getString("password");
            if (userID.equals("admin") && password.equals("password")) {
                return "Successful";
            } else {
                return "Failed";
            }

        } catch (JSONException e) {

            e.getStackTrace();
            return "Failed";

        }
    }
```

- The all the @Path found in your Bluemix Java Application application will be available in the "default" JAX-RS application with a servlet URL pattern of "/api/*".
  - Based on the code for login as shown, the REST service can be invoked by calling the URL  http://<host:port>/api/service/login

- The /login URI POST invocation will trigger checkLogin method. This method does the following actions:
  - Receives the data passed in as creds
  - Parses the user_id and password values Validates the credentials passed

  { “user_id”: “admin”,“password”:”password” }
  - Returns a Response string “Successful”

© 2015 IBM Corporation

# Bluemix Java Application – Create – ServiceAPI.java (/searchtips REST call)

- /searchtips POST RESTful Service

```
@Path("/searchtips")
@POST
public String searchTips(String keyword){
    String QandAList = null;
    try {
        JSONObject keywordObj = new JSONObject(keyword);
        String question = keywordObj.getString("question");
        JSONObject responseJSON = new JSONObject();
        if( question.equals( "depression" ) ) {

            JSONObject questionandasnwer = new JSONObject();
            questionandasnwer.put("question", "What is " + question + "?");
            JSONArray answerArray = new JSONArray();

            JSONObject answers = new JSONObject();
            answers.put( "text", "you can not sleep or you sleep too much");
            answerArray.put( answers );

            answers = new JSONObject();
            answers.put( "text", "you can not concentrate or find that previously easy tasks c
            answerArray.put( answers );

            answers = new JSONObject();
            answers.put( "text", "you feel hopeless and helpless");
            answerArray.put( answers );

            questionandasnwer.put("answers", answerArray);

            JSONArray qanda = new JSONArray();
            qanda.put( questionandasnwer );
            responseJSON.put("qa", qanda);
            return responseJSON.toString();
        } else {
            return "Failed";
        }
    }
    catch (JSONException e) {
        e.printStackTrace();
        System.out.println(e.getMessage());
        return "Failed";
    }
}
```

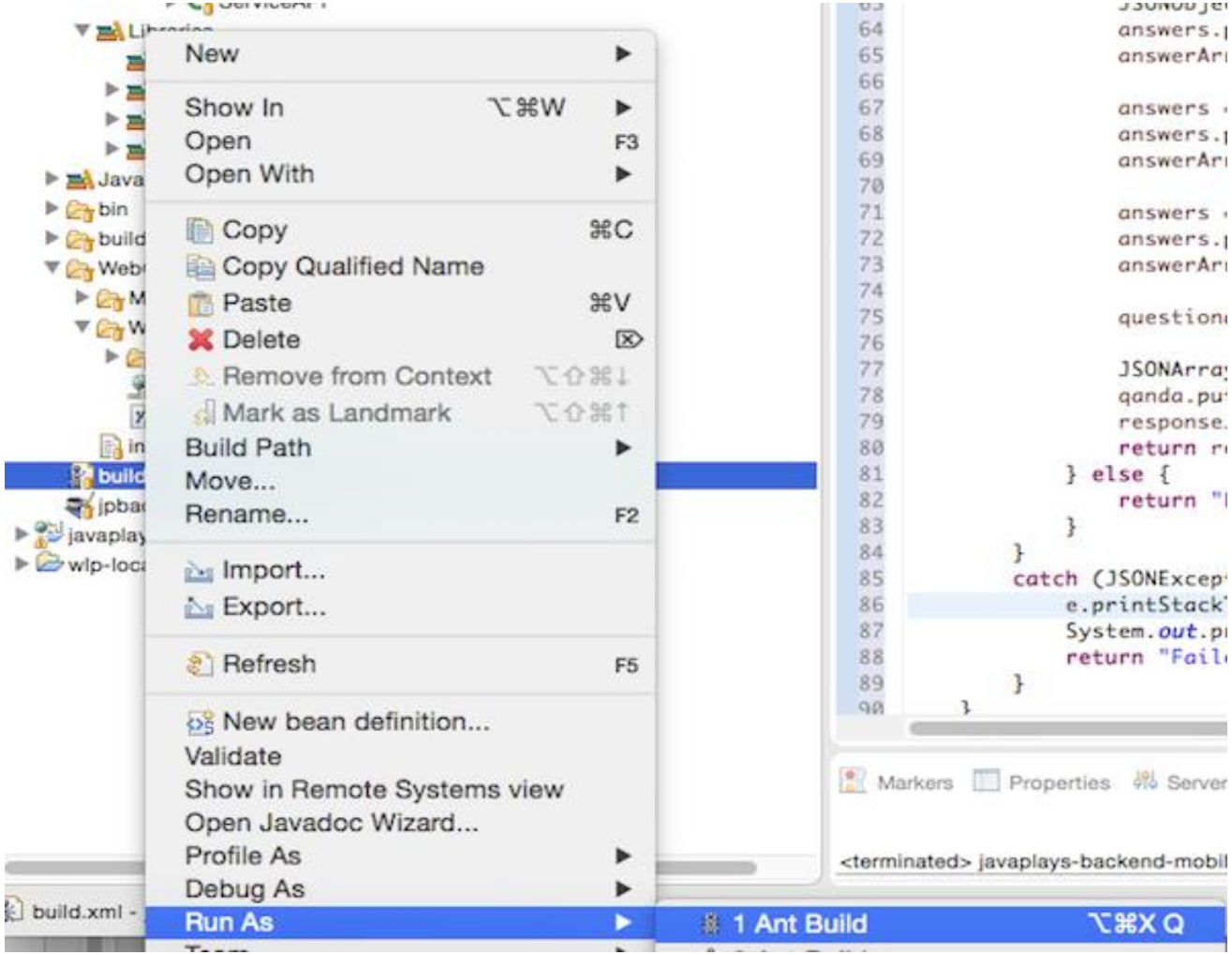
- The all the @Path found in your Bluemix Java Application application will be available in the "default" JAX-RS application with a servlet URL pattern of "/api/*".
  - Based on the code for login as shown, the REST service can be invoked by calling the URL http://<host:port>/api/service/searchtips
- The /searchtips URI POST invocation will trigger searchtips method. This method does the following actions:
  - Receives the data passed in as “keyword”
  - Parse the keyword and retrieve “question”-
  - Validates the question value passed

  { “question” : “depression” }
  - Construct and Return a Response string with all the Answers for a Question

© 2015 IBM Corporation

# Bluemix Java Application – Build

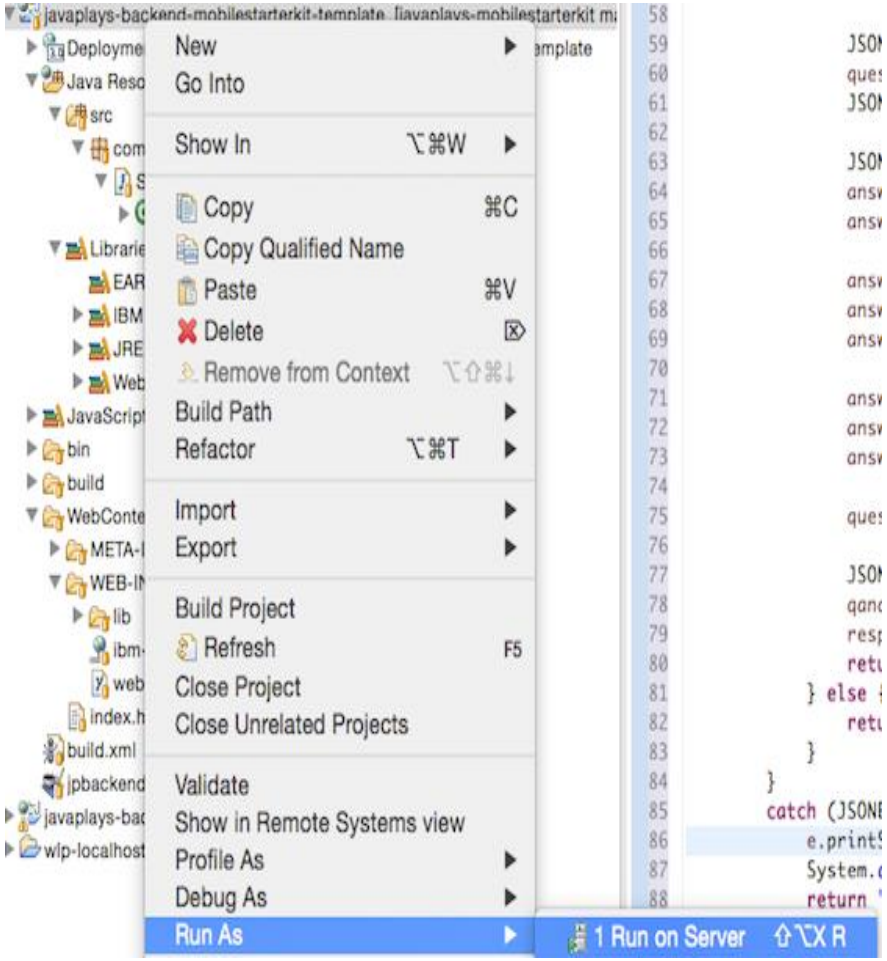
- Browse to root of the project (javaplays-restjava-mobilestarterkit-template/src)→Navigate to build.xml →Right Click →Ant Build

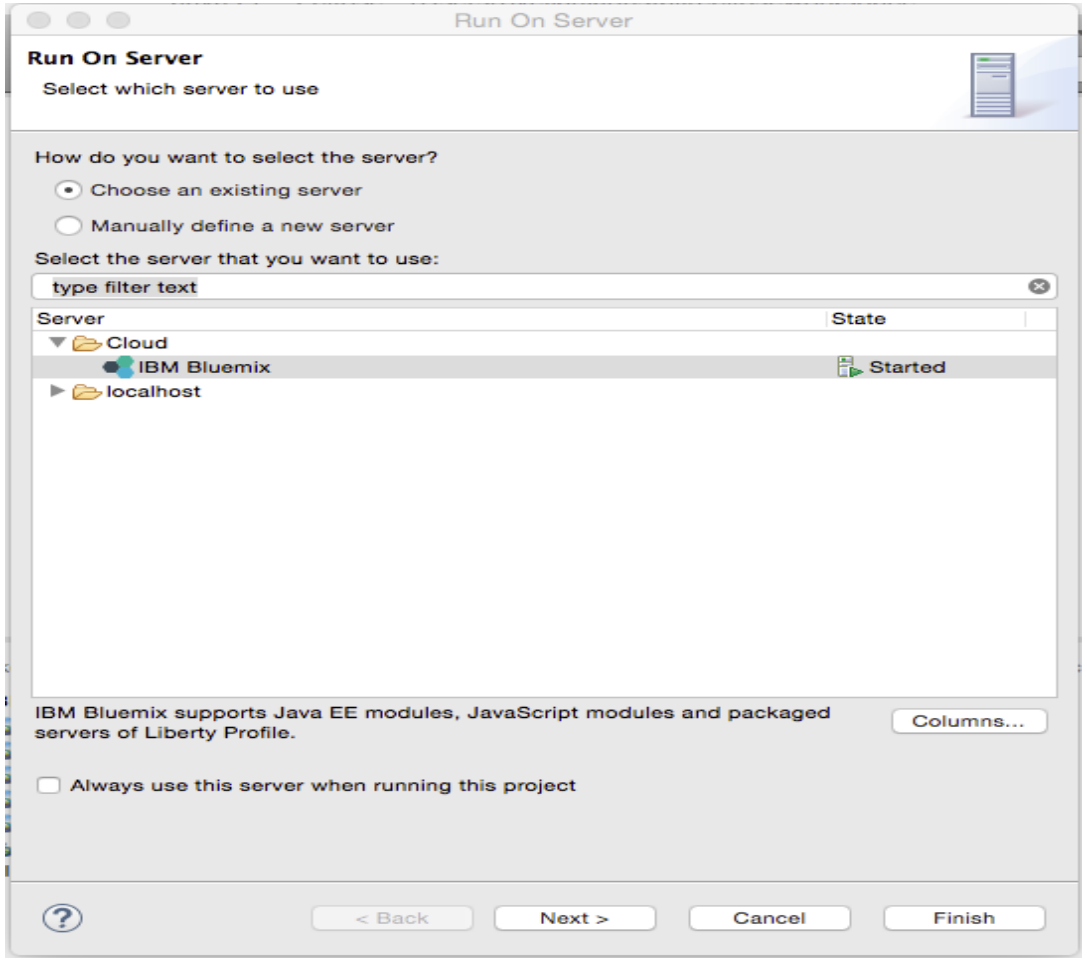



© 2015 IBM Corporation

# Bluemix Java Application – Deploy

- Browse to root of the project ((javaplays-restjava-mobilestarterkit-template/src) →Right Click →Run on Server →Select Bluemix and Deploy

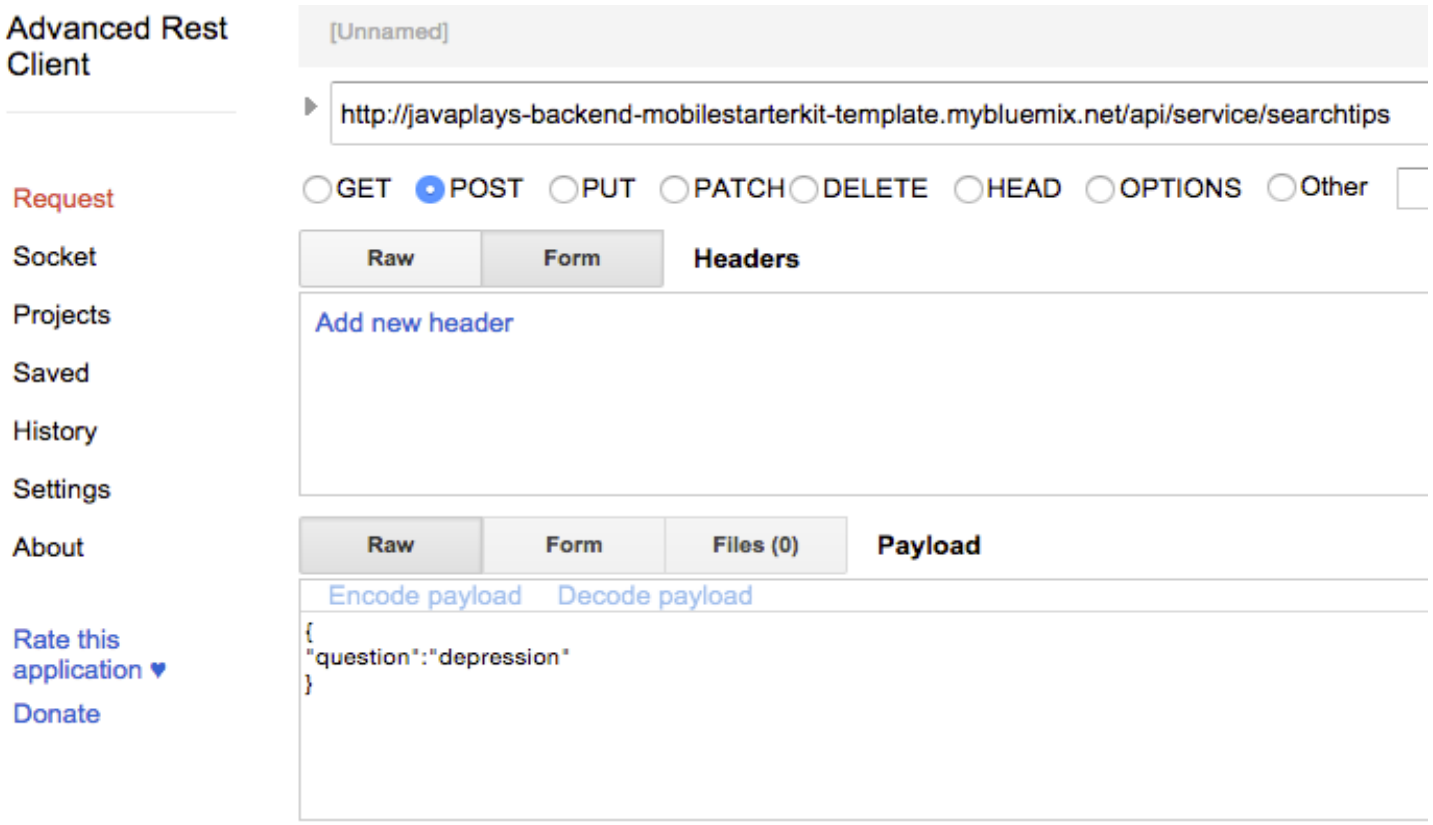



© 2015 IBM Corporation

# Bluemix Java Application – Test

- Chrome Advanced REST Client – Test the REST service




RESULT

- {"qa":[{"answers":[{"text":"you can not sleep or you sleep too much"},{"text":"you can not concentrate or find that previously easy tasks are now difficult"},{"text":"you feel hopeless and helpless"}],"question":"What is depression?"}]}

© 2015 IBM Corporation

# Mobile Application

© 2015 IBM Corporation

# Setup Android Studio

**Android Studio Setup**

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. On top of the capabilities you expect from IntelliJ, Android Studio offers:

Download Android Studio from https://developer.android.com/sdk/installing/index.html?pkg=studio

1. Before you set up Android Studio, be sure you have installed JDK 6 or higher (the JRE alone is not sufficient)—JDK 7 is required when developing for Android 5.0 and higher. To check if you have JDK installed (and which version), open a terminal and type javac -version. If the JDK is not available or the version is lower than 6, go download JDK
2. Launch the .exe file you just downloaded.
3. Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.

Select **Start menu > Computer > System Properties > Advanced System Properties**. Then open **Advanced tab > Environment Variables** and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.7.0_21.

The individual tools and other SDK packages are saved outside the Android Studio application directory. If you need to access the tools directly, use a terminal to navigate to the location where they are installed. For example:
\Users\<user>\sdk\

© 2015 IBM Corporation

# Setup SDK Manager

Android SDK Manager Setup

The Android SDK Manager separates the SDK tools, platforms, and other components into packages for easy access and management. You can also customize which sites the SDK Manager checks for new or updated SDK packages and add-on tools. For example, you can configure the SDK Manager to automatically check for updates and notify you when an installed SDK Tools package is updated. When you receive such a notification, you can then quickly decide whether to download the changes.

By default, Android Studio does not check for Android SDK updates. To enable automatic Android SDK checking:

- Choose **File** > **Settings** > **Appearance & Behavior** > **System Settings** > **Updates**.

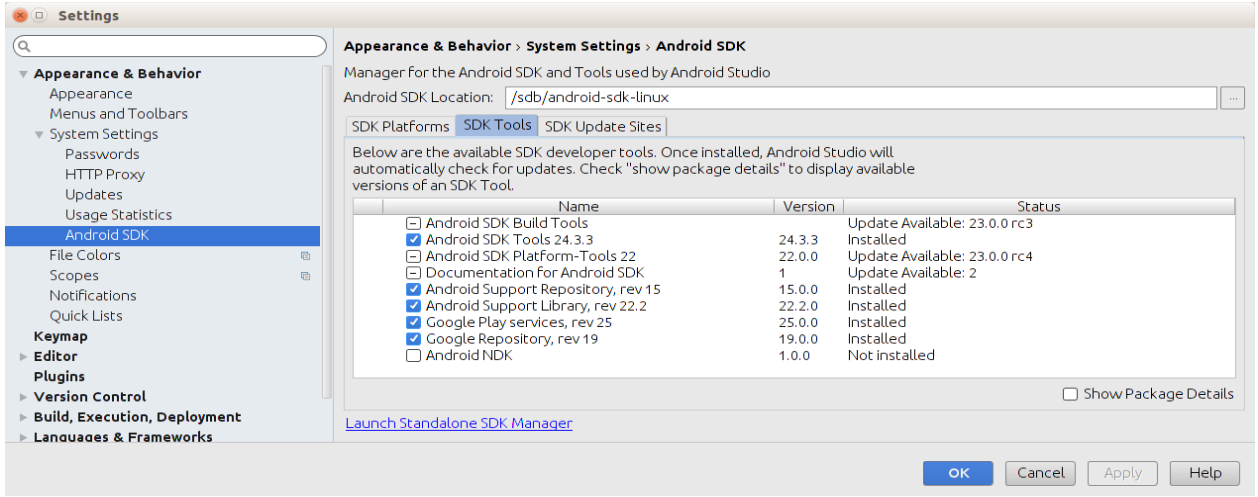
Check the **Automatically check updates for Android SDK** checkbox and select an update channel.

- Click **OK** or **Apply** to enable the update checking.

You can launch the SDK Manager in one of the following ways:

- From the Android Studio **File** menu: **File** > **Settings** > **Appearance & Behavior** > **System Settings** > **Android SDK**.
- From the Android Studio **Tools** menu: **Tools** > **Android** > **SDK Manager**.
- From the SDK Manager icon n in the menu bar.

By default, the SDK Manager instal ( ) l latest packages and tools.
Click the checkbox next to each ad l SDK platform and tool that you want to install. Clear the checkbox to uninstall a SDK platform or tool.
Click **Apply** or **OK** to update the packages and tools




© 2015 IBM Corporation

# Setup AVD Manager

The AVD Manager is a tool you can use to create and manage Android virtual devices (AVDs), which define device configurations for the Android Emulator.
To launch the AVD Manager:
In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager in the Toolbar

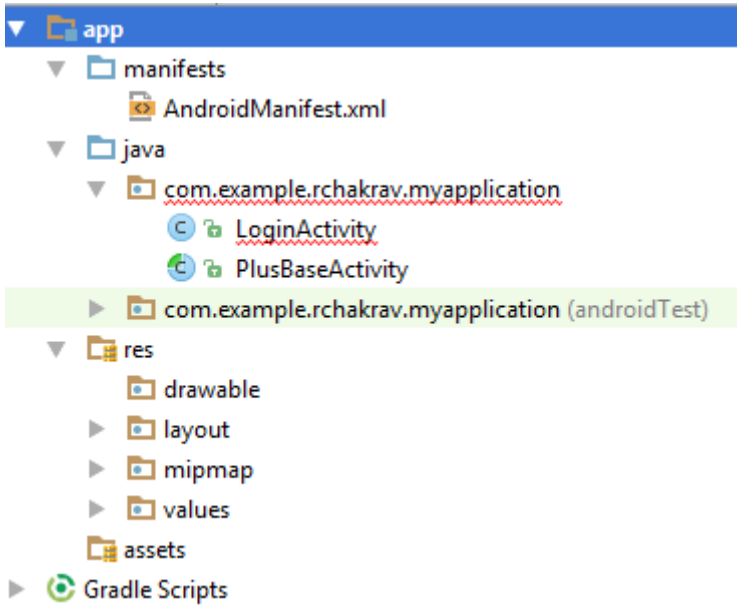
© 2015 IBM Corporation

# Mobile Application - Basics

© 2015 IBM Corporation

# Native Android Application Development

- Anatomy of a native Android Application Project
  - Manifests
    - Contains AndroidManifest.xml file
  - Java
    - Contains Java packages and class definition
  - Res
    - Defines various resources for the application
  - Assets
    - Declares any additional components required by the application
- https://developer.android.com/tools/projects/index.html#ApplicationModules

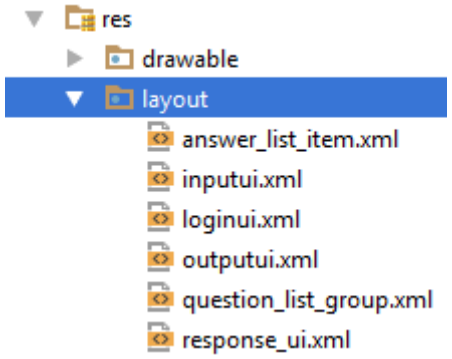



© 2015 IBM Corporation

# Native Android Application Development

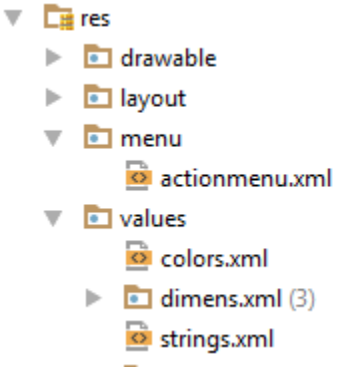
- Principal Concepts and Classes
  - Application
    - Binds everything together and maintains the global status of the application
    - http://developer.android.com/reference/android/app/Application.html
  - Activity
    - User interfaces
    - Each activity focuses on a specific part of the user interaction and several activities form an application
    - http://developer.android.com/reference/android/app/Activity.html

- Designing interfaces
  - Each interface is designed in simple xml format under the project's res->layout section
  - Each interface consists of one or more “Views” (Buttons, TextViews, Editable Textboxes etc.) arranged in specified layouts (Linear, Relative etc.)
    - http://developer.android.com/reference/android/view/View.html
    - http://developer.android.com/guide/topics/ui/declaring-layout.html




© 2015 IBM Corporation

# Native Android Application Development - Contd.




- String constants
  - Instead of using constant values directly in the code, Android prefers defining each string constant as a resource in **res->values->string.xml** with unique ID
  - Once declared as resource, the string is available from anywhere through its unique ID

```
<resources>
    <string name="login_text">Log In</string>
</resource>

<Button
android:text="@string/login_text" >
</Button>
```

- Color constants can be defined in the same way in **res->values->color.xml** and accessed from anywhere using the ID
- More about Resources : http://developer.android.com/guide/topics/resources/index.html

© 2015 IBM Corporation

# Native Android Application Development - Contd.

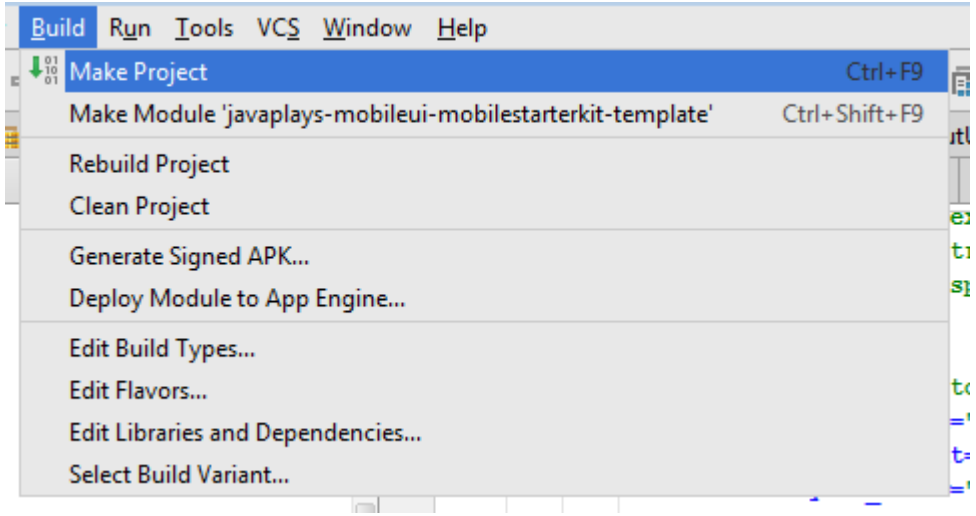
- Putting things together
  - AndroidMenifest.xml contains essential information required to build and run the application
    - Defines specific SDK to use, minimum target Android version, access permissions required (Internet, sensors, Phonebook etc.)
    - Defines the main application class
    - Registers each of the activities
    - Identifies the “launching” activity
    - http://developer.android.com/guide/topics/manifest/manifest-intro.html

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ibm.startkit.android.app"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="18" />
    <!-- Permissions -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <application
        android:name="com.ibm.startkit.android.app.BlueMobileApp"
        android:allowBackup="true"
        android:icon="@drawable/logo"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="com.ibm.startkit.android.view.apptasks.InputUI"
            android:label="@string/app_name"
            android:configChanges="orientation|screenSize">
        </activity>
        <activity
            android:name="com.ibm.startkit.android.view.apptasks.OutputUI"
            android:label="@string/app_name"
            android:configChanges="orientation|screenSize">
        </activity>
        <activity
            android:name="com.ibm.startkit.android.view.LoginUI"
            android:label="@string/app_name"
            android:configChanges="orientation|screenSize">
           <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

© 2015 IBM Corporation

# Native Android Application Development - Contd.

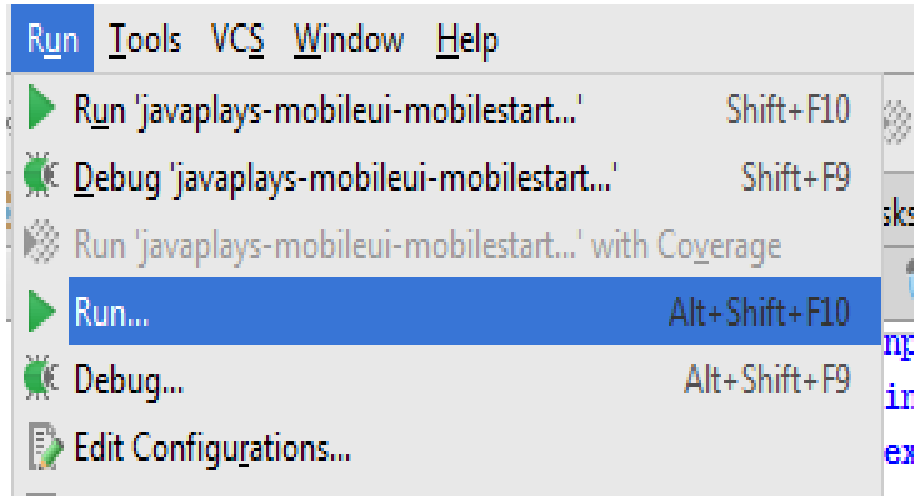
- Building and running Android Application

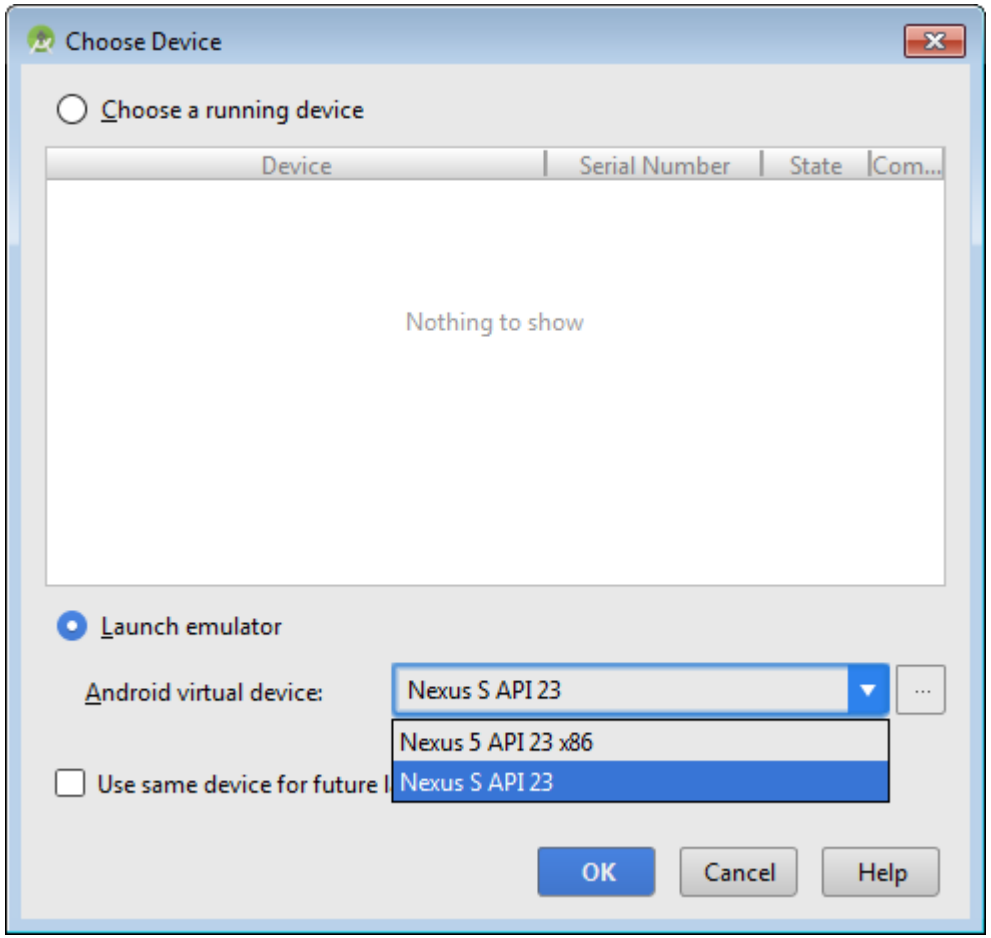



© 2015 IBM Corporation

# Native Android Application Development - Contd.

- Running Android Application




© 2015 IBM Corporation

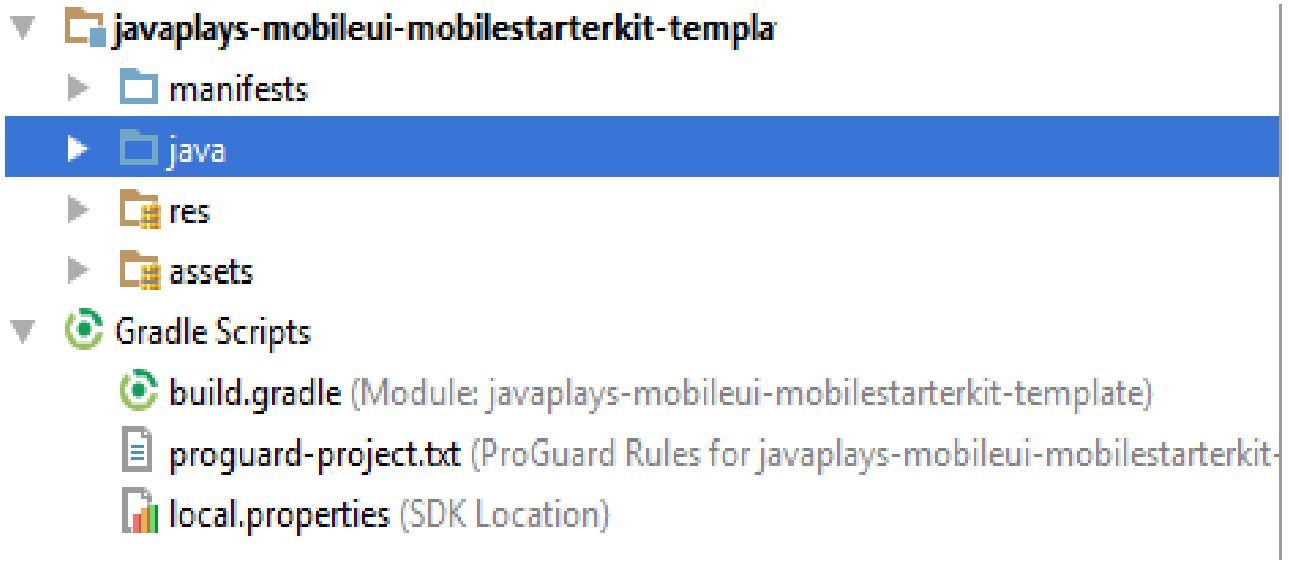
# Mobile Application - Create

© 2015 IBM Corporation

# Let us develop something now...

Get the Code:

- Option1 –
  - Get the Code and Android Studio Installable from the Instructor
- Option 2 -
  - Download the template from Git
    - https://hub.jazz.net/git/ecosysdevcnc/javaplays-mobilestarterkit/
      - Javaplays-mobileui-mobilestarterkit-template
        - Unzip to folder
      - Javaplays-mobileui-mobilestarterkit-templatecode
  - Open the project in Android Studio
    - Select the Option → Open Existing Project → Select the Unzipped folder of the template → Javaplays-mobileui-mobilestarterkit-template
    - Also open Javaplays-mobileui-mobilestarterkit-templatecode folder files in another editor

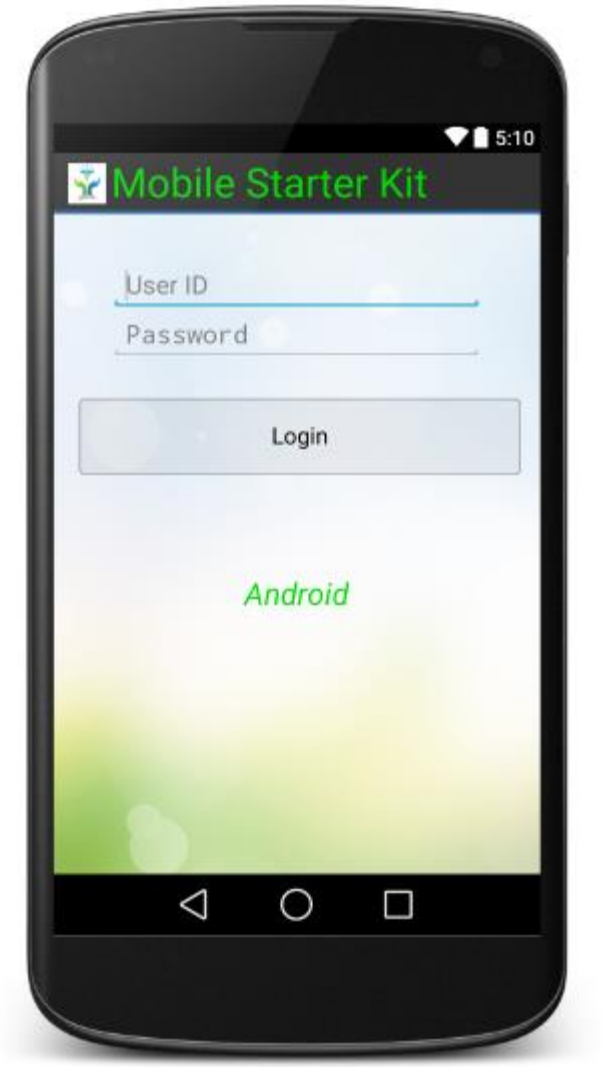



© 2015 IBM Corporation

# Make the Login Interface

- Step 1 : Design the interface
  - Copy the code from **loginui.xml.code** to projects **res->layout->loginui.xml**
- Step 2 : Set the content of the UI in LoginUI activity
  - In the project's **java->com->ibm->startkit.android->view->LoginUI.java**, locate the onCreate() event and uncomment the necessary code.
- Step 3 : Add some event to the interface's button even
  - In the project's **java->com->ibm->startkit.android->view->LoginUI.java**, locate the setButtonEvent() function
  - Copy the code from **loginui.button_event.code** into the setButtonEvent() function
- Step 4 : Execute an asynchronous task to check the user's validity
  - In the project's **java->com->ibm->startkit.android->view->LoginUI.java**, locate the checkLogin() function
  - Copy the code from **loginui.checklogin.code** into the checkLogin() function

- At this stage, we have an UI that calls an Aynchronous task but the task itself is not ready




© 2015 IBM Corporation

# Define the Asynchornous Task

- Step 1 : Define the task
  - In the project's **java->com->ibm->startkit.android->bluemix->LoginTask.java**, locate the performTask() function
  - Copy the code from **logintask.perform_task.code** into performTask() function
  - Do NOT forget to remove the “return null” from the function

- At this stage, we have an UI that calls an Aynchronous task and the task can execute a server side via REST API

© 2015 IBM Corporation

# Receive the task when complete

- Step 1 : Receive the task and take necessary steps
  - In the project's **java->com->ibm->startkit.android->view->LoginUI.java**, locate the receiveResult() function
  - Copy the code from loginui.receive_result.code into this function

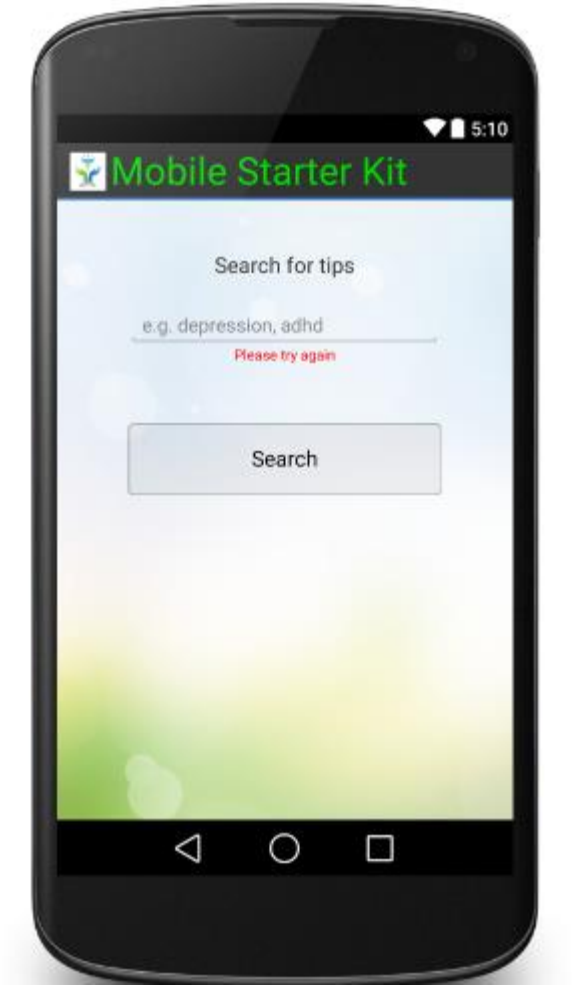
- At this stage, we have a working UI that can execute a REST API via asynchronous task and take necessary action once the task in complete

© 2015 IBM Corporation

# Make the Search Interface

- Step 1 : Design the interface
  - Copy the code from **searchui.xml.code** to projects **res->layout->searchui.xml**
- Step 2 : Set the content of the UI in LoginUI activity
  - In the project's **java->com->ibm->startkit.android->view->apptasks->SearchUI.java**, locate the onCreate() event and uncomment the necessary code.
- Step 3 : Prepare the UI components and add some event to the interface's button even
  - In the project's **java->com->ibm->startkit.android->view->apptasks->SearchUI.java**, locate the setupUIComponents() function
  - Copy the code from **searchui.setupUI.code** into the setupUIComponents() function
- Step 4 : Execute an asynchronous task to search for some tips
  - In the project's **java->com->ibm->startkit.android->view->apptask->SearchUI.java**, locate the doSearch() function
  - Copy the code from **searchui.do_search.code** into the doSearch() function

- At this stage, we have an UI that calls an Aynchronous task but the task itself is not ready

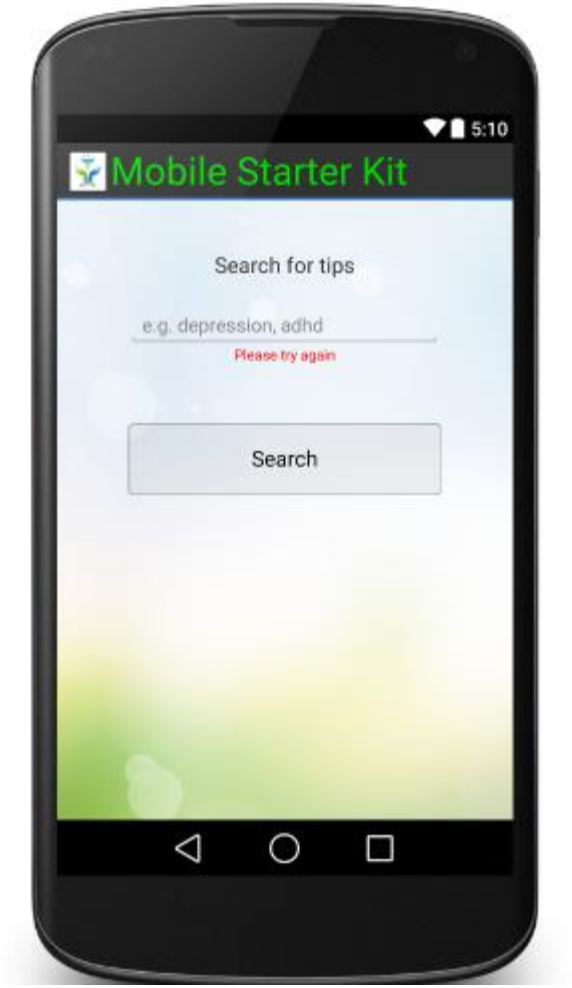



© 2015 IBM Corporation

# Define the Asynchronous task

- Step 1 : Define the task
  - In the project's **java->com->ibm->startkit.android->bluemix->SearchTipsTask.java**, locate the performTask() function
- Step 2 : Add some code to perform the remote API task
  - Copy the code from **searchtipstask.perform_task.code** into performTask() function
  - Do NOT forget to remove the “return null” from the function

- At this stage, we have an interface that calls an Aynchronous task and the task can execute a server side via REST API

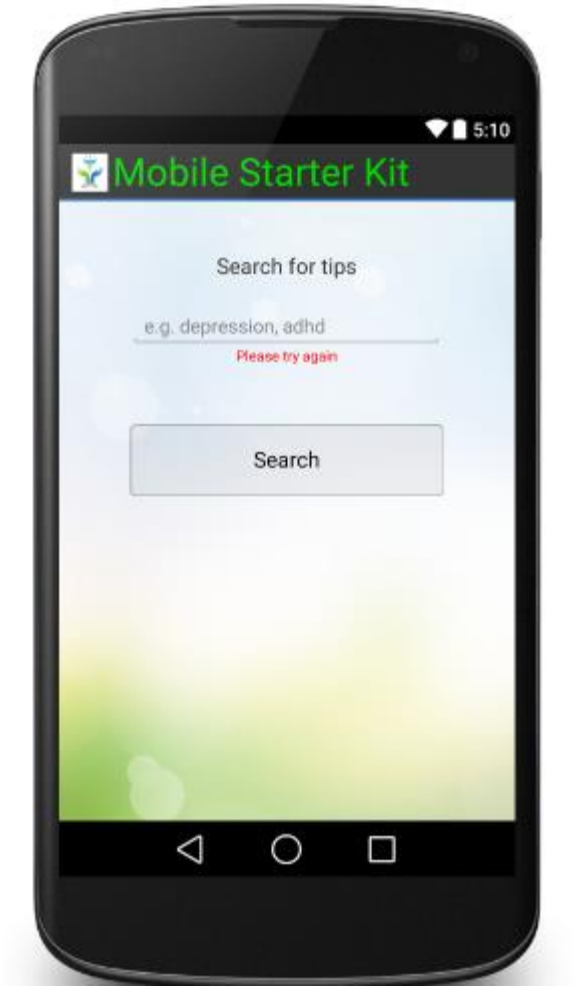



© 2015 IBM Corporation

# Receive the task when complete

- Step 1 : Receive the task and take necessary steps
  - In the project's **java->com->ibm->startkit.android->view->apptasks->SearchUI.java**, locate the receiveResult() function
  - Copy the code from **searchui.receive_result.code** into this function

- At this stage, we have a working UI that can execute a REST API via asynchronous task and take necessary action once the task in complete

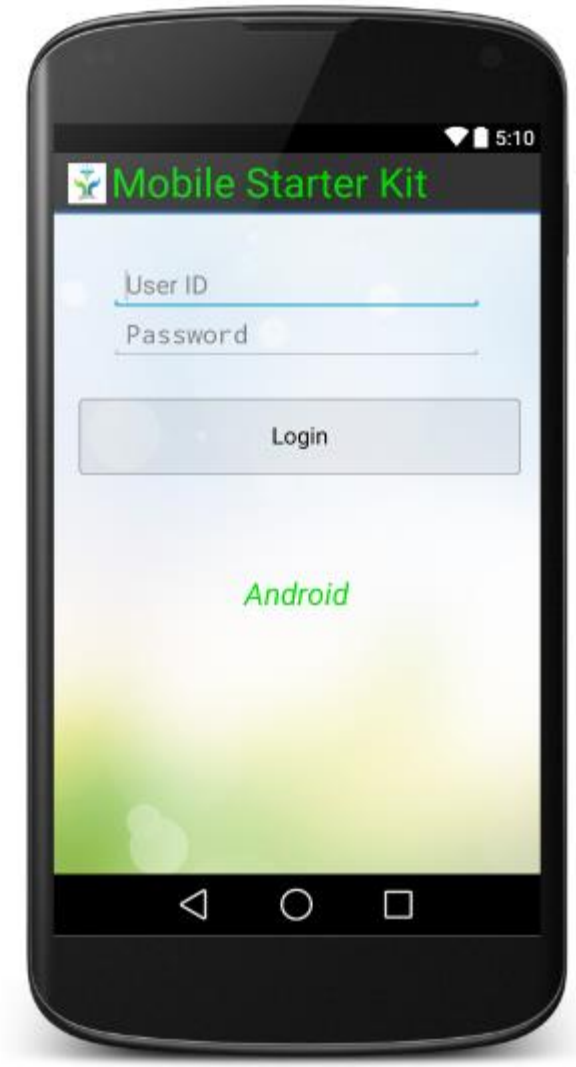



© 2015 IBM Corporation

# Registering the interfaces

- Step 1 : Manifests->AndroidManifest.xml
  - Copy the code from **androidmanifest.activity_registration.code** within the application tag

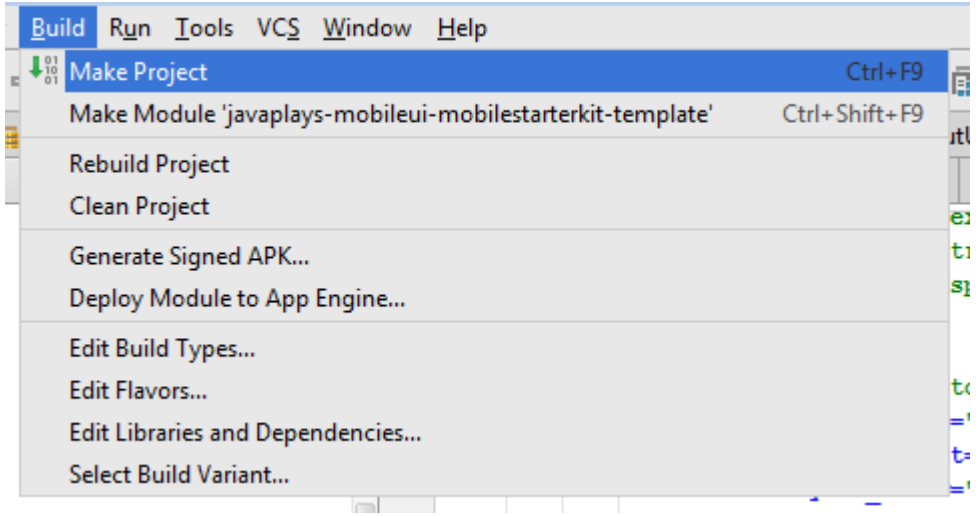
- At this stage, we have a working application ready to be built and run
  - Confirm the “baseuri” value is correct in assets->starterkit.properties file




© 2015 IBM Corporation

# Build the Android Application

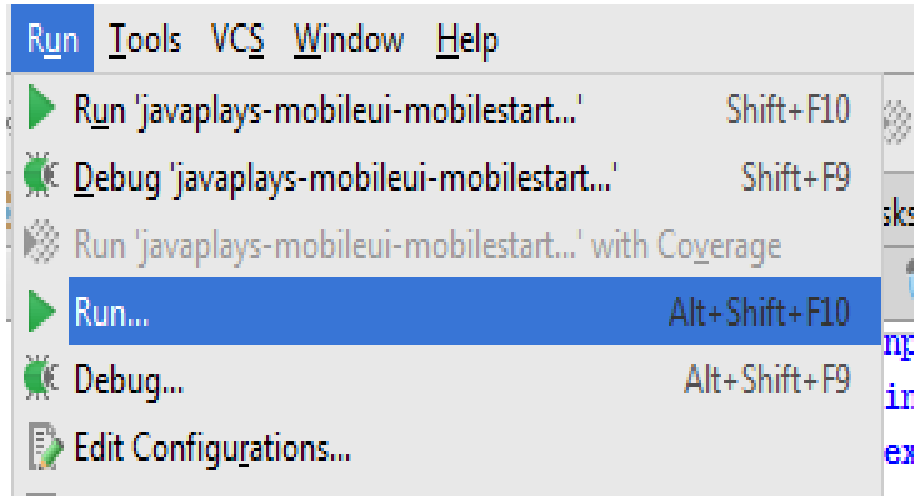
- Building and running Android Application

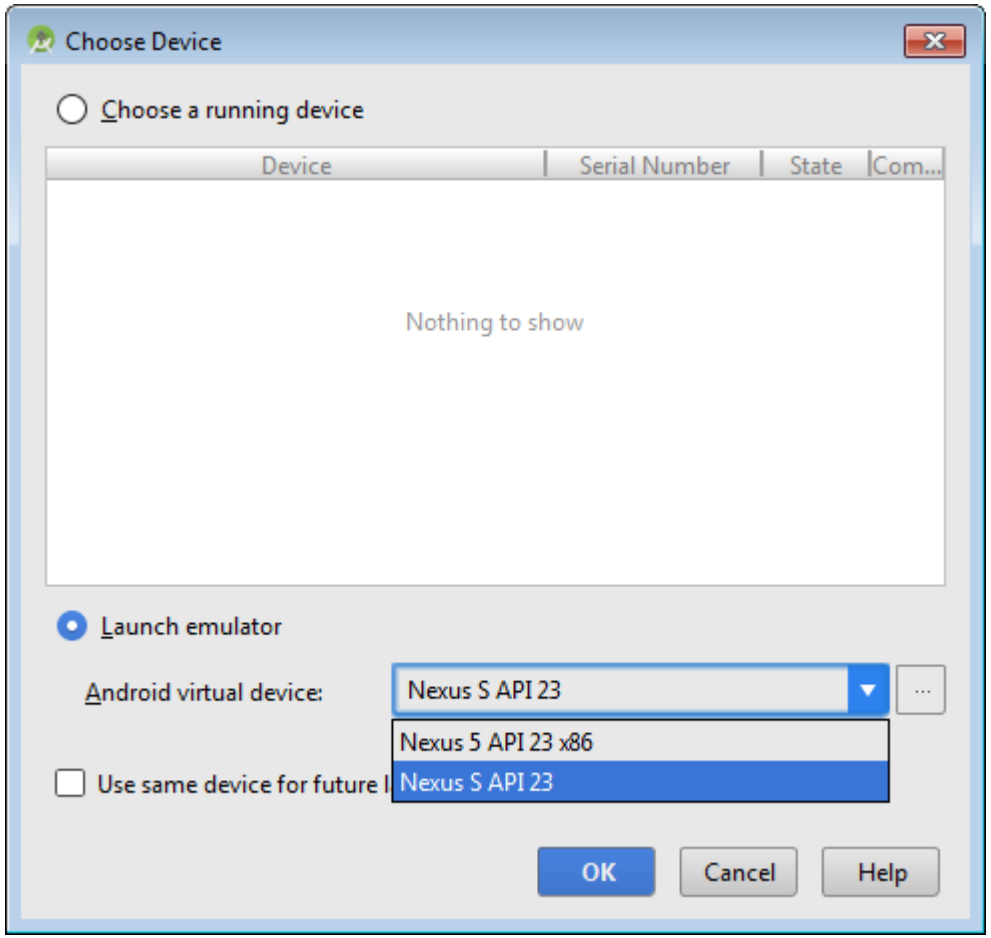



© 2015 IBM Corporation

# Run the Android Application

- Running Android Application

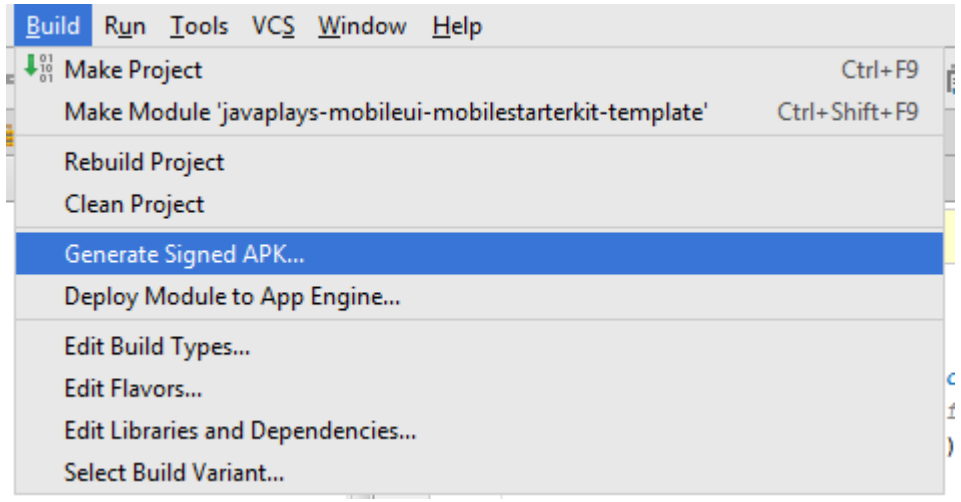


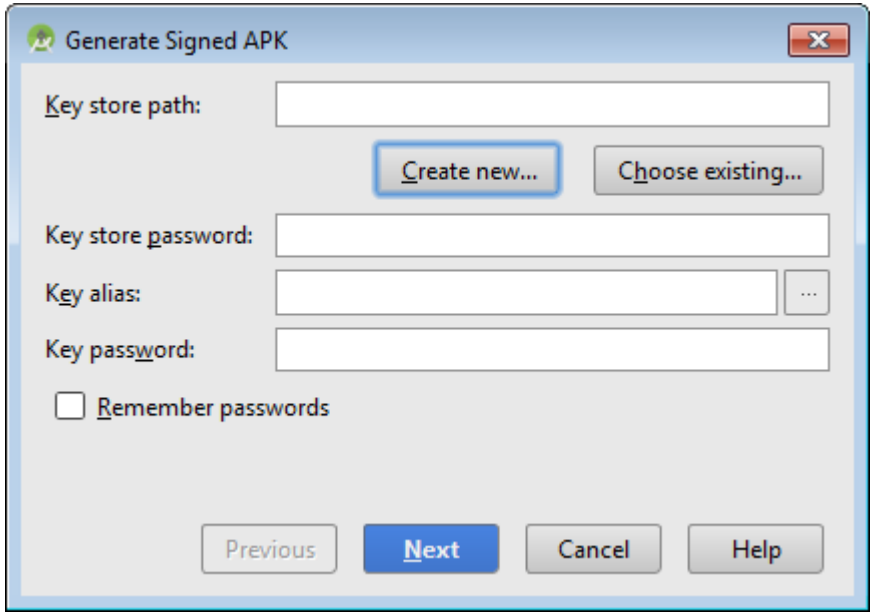




© 2015 IBM Corporation

# Generating an APK  Signing an Android Application

- From **Build** menu choose **"Generate Signed APK**
- From the dialogue box, click **Create New**
  - If you already have a keystore, please choose the **Choose Existing** and provide the necessary password

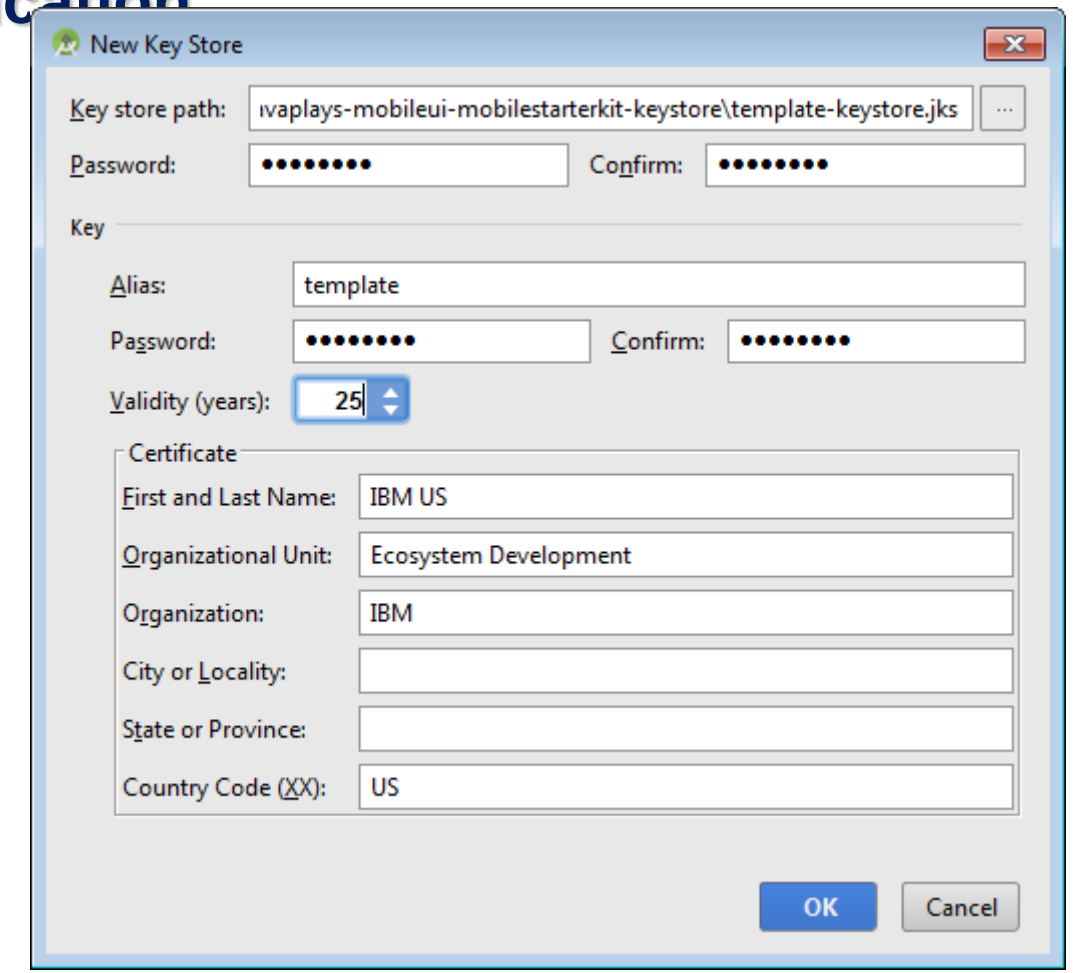



© 2015 IBM Corporation

# Generating an APK  Signing an Android Application

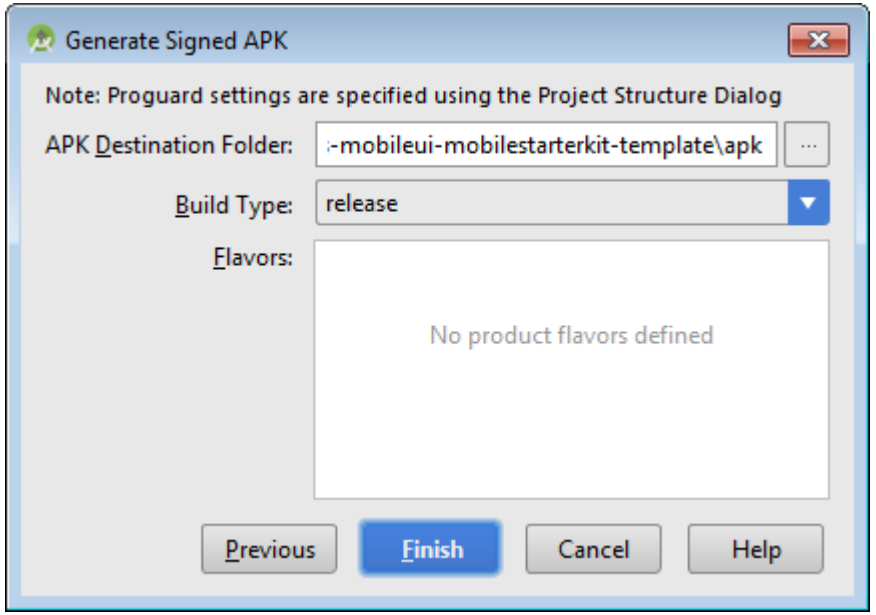
- Provide the Key Store Path
- Provide the key
  - In this example every password is “template”




© 2015 IBM Corporation

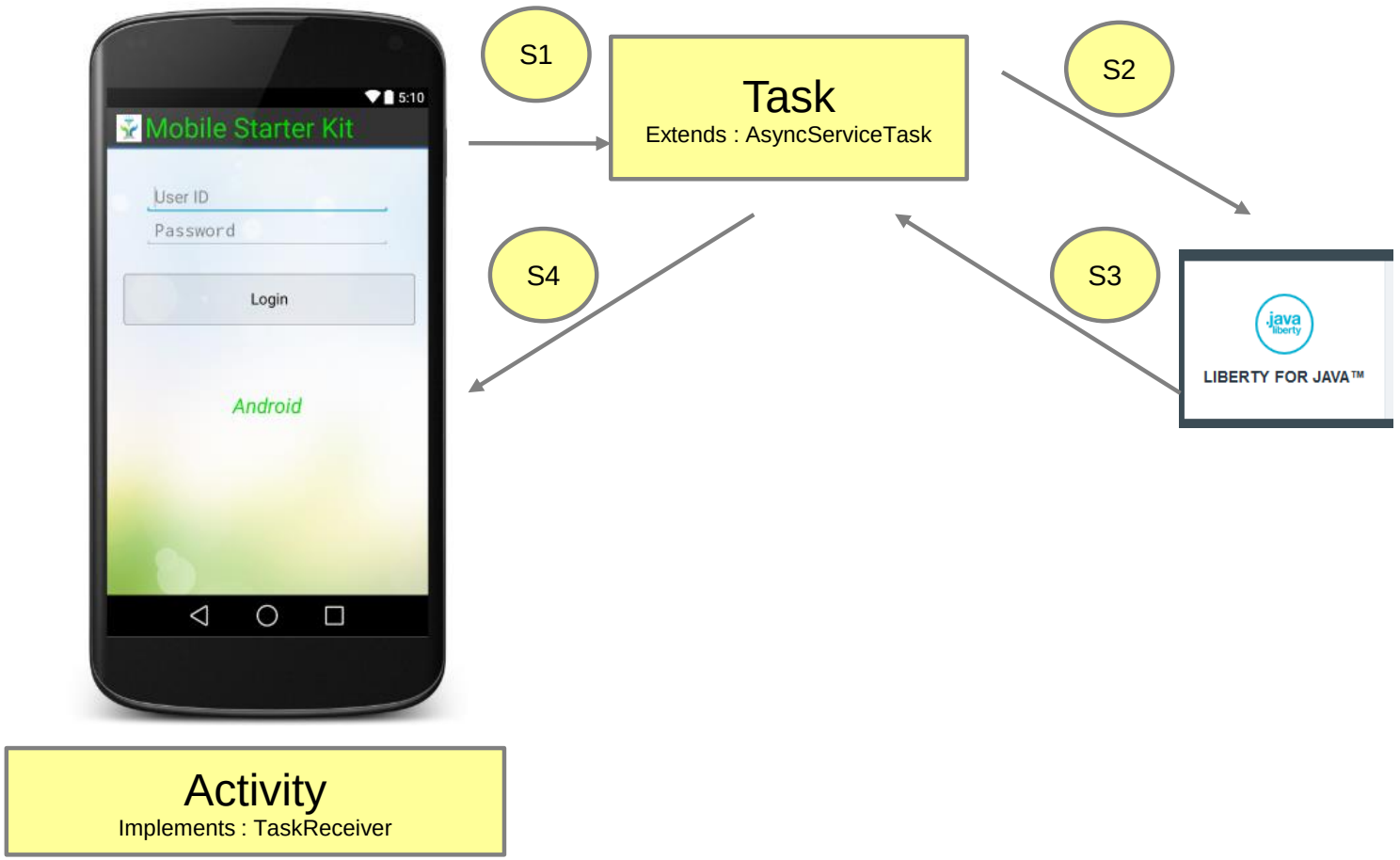
# Signing an Android Application

- Once successful, the wizard will ask for a destination
- Click Finish
- The process will confirm the generation of the APK
  - Now your application is ready to be released




© 2015 IBM Corporation

# Task Execution Protocol – Step by step




- S1 :
  - Activity creates the task and assigns an ID to it
  - Activity registers itself as a listener to the postExecution event of the task
  - Activity starts the task in a separate thread
- S2 :
  - Task executes a POST method and waits for the result
- S3 :
  - The response is received by the task thread
  - Task raises a postExecute() event and passes the response to any listener
- S4 :
  - Activity responds to the event and process the result

© 2015 IBM Corporation

# Task Execution Protocol – Step by step

```
//SearchUI.java - if the user clicks the button
{

...
SearchTipsTask stt = new SearchTipsTask( ... );
stt.setReceiver(this);
stt.execute(_seeds);
...

}
```

S1

```
//AsyncServiceTask.java
@Override
protected String doInBackground( String... params )
{

//performTask is implemented by the
//SearchTipsTask class
return performTask( params );
}
```

S2

LIBERTY FOR JAVA™

S3

```
//AsyncServiceTask.java
@Override
protected void onPostExecute( String result ){
   _response = result;
   if( _tr != null ){
      //calls the receiver's receiveResult function
      //adds the serviceID here
      //in this case the receiver is SearchUI.java
_tr.receiveResult( _response,_serviceID);
}
   }
}
```

S4

```
@Override
public void receiveResult( String response,
                           String source ) {
{

 //make sure the task ID is the search task
 if( source.equals( correct_id ) ){
//perhaps do something with the response
 }
}
```

© 2015 IBM Corporation

# How are all these stitched together? - without headache

- com.ibm.android.asynctask.TaskReceiver - Interface
  - Every activity that required to use a REST API implements this class
  - This class enforces the activities to declare a receiveResult function

```
public interface TaskReceiver{
public void receiveResult( String response, String source );
}
```

String response

- The response received from REST call

String Source

- The ID of the asynchronous task generating the response

```
//Example : SearchUI.java
public void receiveResult( String response, String source ) {

//make sure the task ID is the search task
if( source.equals( SearchTipsTask._SEARCH_DOC_ID ) ){

        //do something with the response
}
}
```

© 2015 IBM Corporation

# How are all these stitched together? - without headache

- com.ibm.android.asynctask.AsyncServiceTask - Abstract
  - Every task that calls REST API extends this class
  - This class enforces the tasks to define a REST API call

```
public abstract String performTask( String... params)
        throws IOException, JSONException, URISyntaxException;
```

String... params

- Input parameter needed for the task

```
//Example : SearchTipsTask
@Override
protected String performTask(String... params) throws IOException,
                        JSONException, URISyntaxException {

//retrieve the parameter
String keyword = params[0];

//POST and return the response
return performPost( ... );
}
```

© 2015 IBM Corporation

# Questions?

© 2015 IBM Corporation