Mobile Application Development: Lab 2

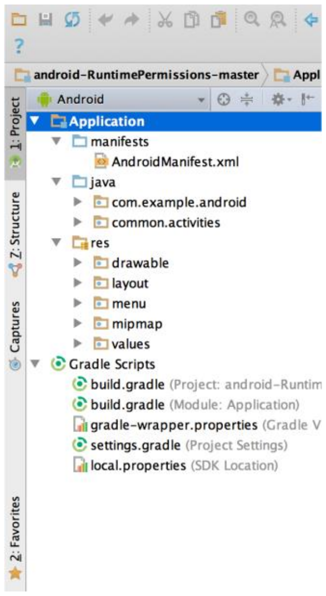
## Objectives

- Understanding Android Studio
- Understanding Android Templates
- Practice Activities

**Objective 1: Understanding Android Studio**

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Environment


android-RuntimePermissions-master
Appl
Android
1: Project
Z: Structure
Captures
2: Favorites
Application
manifests
AndroidManifest.xml
java
com.example.android
common.activities
res
drawable
layout
menu
mipmap
values
Gradle Scripts
build.gradle (Project: android-Runtim
build.gradle (Module: Application)
gradle-wrapper.properties (Gradle V
settings.gradle (Project Settings)
local.properties (SDK Location)

## Project Structure:

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select**Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems**view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

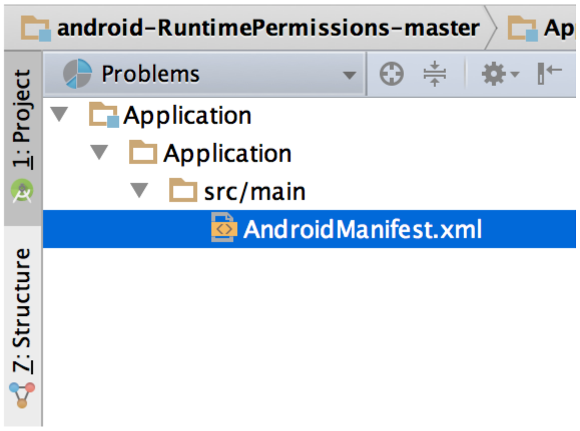
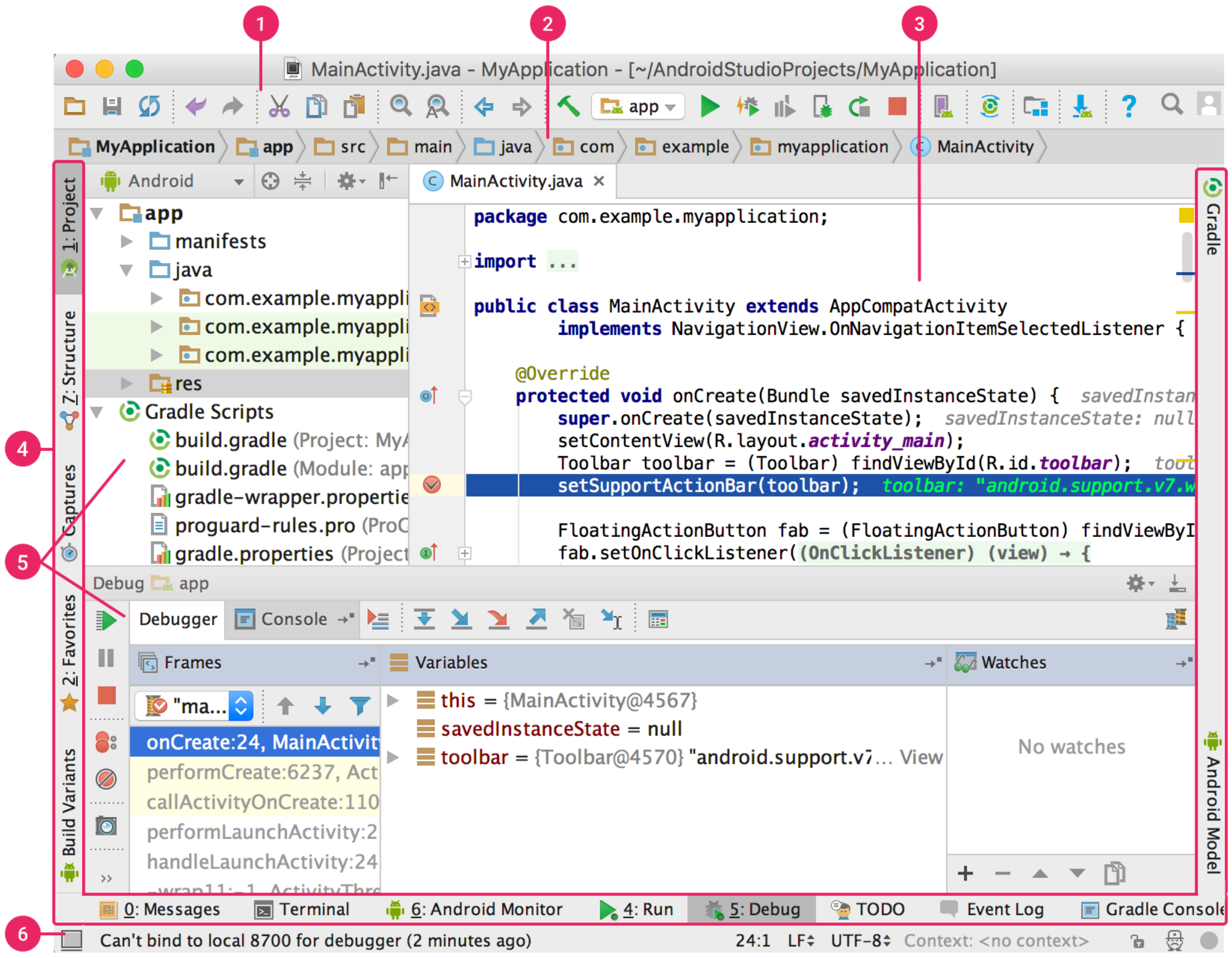


**Figure 2.** The project files in Problems view, showing a layout file with a problem

## Android Studio Lookout:



```
package com.example.myapplication;

import ...

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {   savedInstan
        super.onCreate(savedInstanceState);   savedInstanceState: null
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);   tool
        setSupportActionBar(toolbar);   toolbar: "android.support.v7.w

        FloatingActionButton fab = (FloatingActionButton) findViewByI
        fab.setOnClickListener((OnClickListener) (view) → {
```

```
onCreate:24, MainActivit
performCreate:6237, Act
callActivityOnCreate:110
performLaunchActivity:2
handleLaunchActivity:24
```

```
this = {MainActivity@4567}
savedInstanceState = null
toolbar = {Toolbar@4570} "android.support.v7... View
```

```
Can't bind to local 8700 for debugger (2 minutes ago)        24:1   LF÷   UTF-8÷   Context: <no context>
```

**Figure 3.** The Android Studio main window.

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

## Objective 2: Understanding Android Live Templates

- Live Templates are pre-built code chunks/ lines of code used for rapid application development.
- One can use already created templates or can also create their own templates.

⇨ **Inserting Live templates**

Inserting live templates is the easiest thing in android, android always does this by itself, whenever you write the code, just pressing the tab will automatically insert the related chunk of code.

- To Browse the current Live Templates:
- Go to **File > Settings > Editor > Live Templates**.

⇨ **Make you own Live Template**

- Just write the code, highlight it, and go to Tools > Save as Live Template.

**Code completion**

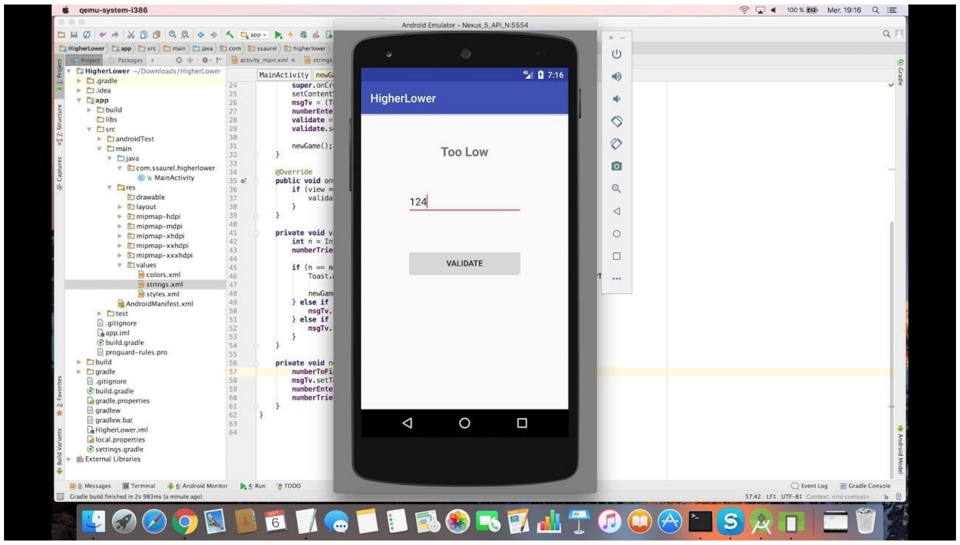
**Table 2.** Keyboard shortcuts for code completion.

| Type | Description | Windows and Linux | Mac |
|---|---|---|---|
| Basic Completion | Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members. | Control+Space | Control+Space |
| Smart Completion | Displays relevant options based on the context. Smart completion is aware of the expected type and data flows. If you call Smart Completion twice in a row, you see more results, including chains. | Control+Shift+Space | Control+Shift+Space |
| Statement Completion | Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc. | Control+Shift+Enter | Shift+Command+Enter |

## Objective 3: Practice Activities

Actvitiy 1:

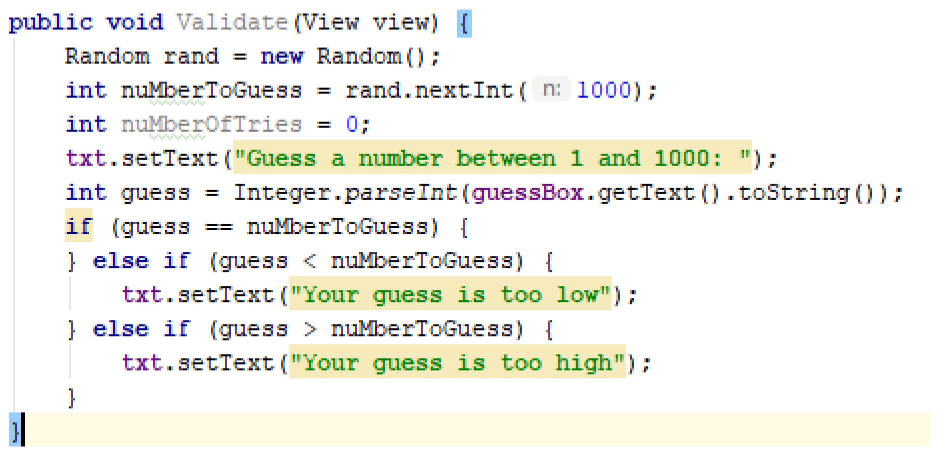
Random number guessing game: The computer thinks of a number from 1 to 1000. The user makes guesses, and after each incorrect guess, the app hints to the user whether the right answer is higher or lower than their guess. The game ends when the player guesses the right number.



The components to add

| View | Value | Event |
|---|---|---|
| TextView | (Too Low, Too High or You Guessess Correctly) | None |
| EditText | | |
| Button | Validate | Validate() |

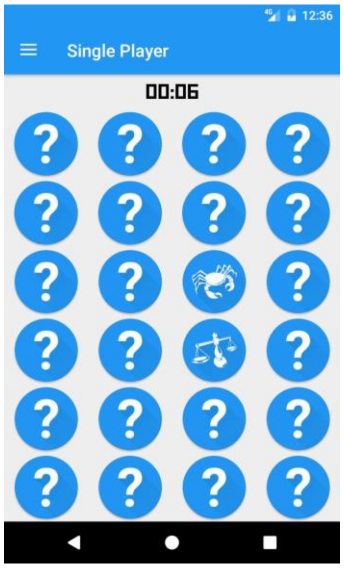
Hint: Following Code



```
public void Validate(View view) {
    Random rand = new Random();
    int nuMberToGuess = rand.nextInt( n: 1000);
    int nuMberOfTries = 0;
    txt.setText("Guess a number between 1 and 1000: ");
    int guess = Integer.parseInt(guessBox.getText().toString());
    if (guess == nuMberToGuess) {
    } else if (guess < nuMberToGuess) {
        txt.setText("Your guess is too low");
    } else if (guess > nuMberToGuess) {
        txt.setText("Your guess is too high");
    }
}
```

Activity 2:
Memory Game: Several buttons are shown on screen, "face down", and the user needs to try to find pairs that match up when flipped over.

**Hint:** You can use arrays to store the values of the images as Adjacency Array.

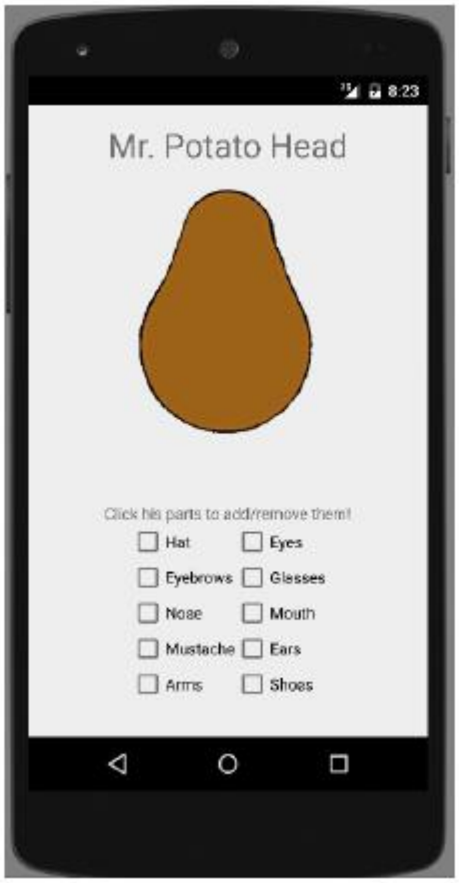


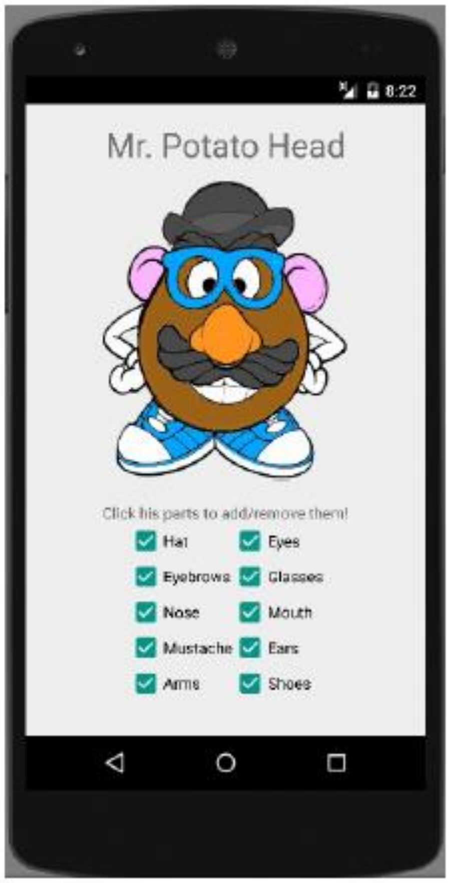
Download Icons Freely : https://www.flaticon.com/packs

https://www.iconfinder.com/search/?q=image

## Activity 3:

***Mr. Potato Head*** *(thanks to Victoria Kirst for original assignment idea and images!)* Write an app that displays a "Mr. Potato Head" toy on the screen as an `ImageView`. The toy has several accessories and body parts that can be placed on it, such as eyes, nose, mouth, ears, hat, shoes, and so on. We will provide you with image files for each body part and accessory, such as **body.png**, **ears.png**, **hat.png**, and so on. Initially your image view should display only the toy's body, but if the user checks/unchecks any of the check boxes below the toy, the corresponding body part or accessory should appear/disappear. The way to display the various body parts is to create a separate `ImageView` for each part, and lay them out in the XML so that they are superimposed on top of each other. You can achieve this with a `RelativeLayout` in which you give every image the same position, though you should probably nest it in some other overall layout for the screen. The check boxes should align themselves into a grid of rows and columns. You can set whether or not an image (or any other widget) is visible on the screen by setting its `android:visibility` property in the XML, and/or by calling its `setVisibility` method in your Java code. The `setVisibility` method accepts a parameter such as `View.VISIBLE` or `View.INVISIBLE`. There is also a `getVisibility` method if you need to check whether a widget is currently visible.


8:23
Mr. Potato Head
Click his parts to add/remove them!
☐ Hat ☐ Eyes
☐ Eyebrows ☐ Glasses
☐ Nose ☐ Mouth
☐ Mustache ☐ Ears
☐ Arms ☐ Shoes


8:22
Mr. Potato Head
Click his parts to add/remove them!
✓ Hat ✓ Eyes
✓ Eyebrows ✓ Glasses
✓ Nose ✓ Mouth
✓ Mustache ✓ Ears
✓ Arms ✓ Shoes