EECS1022 (M,N,O) Winter 2021
Lab7 (Mobile Computing Part 1)
Building a Bank App using Android Studio

Chen-Wei Wang

Release Date: Friday, March 12
Due Date: 23:59 EST, Friday, March 26

- The format of this lab (to be completed in Android Studio) is different from those of all previous labs (completed in Eclipse). Read the instructions carefully.
- You are required to **work on your own** for this lab. **No** group partners are allowed.

  Plagiarism checks will be run on all submissions, and suspiciously similar ones will be reported to Lassonde.
- To complete this lab, it is **strictly forbidden** for you to use any library class (e.g., `ArrayList`, `Arrays`). Violating this requirement will cause a **50% penalty** on your lab marks.
- For this lab, you will be graded not only by JUnit tests given to you, **but also additional tests** covering some other input values. This is to encourage you to take more responsibility for the correctness of your code, by writing your own tests.
- Your lab assignment is **not** graded during the weekly scheduled lab sessions.
  - Follow the instructions to submit (via eClass) the required file(s) for grading.
  - Emailing your solutions to the instructor or TAs will **not** be acceptable.
- Texts in blue are hyperlinks to the corresponding documents/recordings.

Policies

- **Your (submitted or un-submitted) solution to this lab exercise (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.**
- When you submit your lab, you claim that it is **solely** your work. Therefore, it is considered as **a violation of academic integrity** if you copy or share **any** parts of your Java code during **any** stages of your development.
- When assessing your submission, the instructor and TA may examine your code, and suspicious submissions will be reported to the department if necessary. **We do not tolerate academic dishonesty**, so please obey this policy strictly.
- You are entirely responsible for making your submission to the TA in time. Back up your work **periodically**, so as to minimize the damage should any sort of computer failures occur. Follow this tutorial series on setting up a **private** Github repository for your Java projects.
- The deadline is **strict** with no excuses. Refer to the course syllabus for the policy on a late submission.

Contents

Learning Outcomes

By completing the assigned exercises of this lab, you are expected to be able to:

1. Apply the Model-View-Controller (MVC) Pattern.
2. Design and Implement a GUI for a Bank App.
3. Practice Event-Driven Programming:
   - Spinner (Drop-Down Menu List)
   - Text View
   - Edit Text Field
   - Button
4. Practice the pattern of controller: multiple buttons that access and modify a single, shared object (i.e., `Bank` object).
5. Attach Controller Methods to GUI Components
6. Implement model classes:
   - multiple model classes: attributes in a model class are of array-referenced types.
   - local variable declarations
   - variable assignments
   - object creations
   - method calls using the dot notation

Assumptions

- You have already setup a Github account and stored work in a **private** repository: `EECS1022-W21-workspace`.

  Note. You only submit your lab through eClass, not Github. Though not required, it is highly recommended that you adapt to the practice of backing your work using a versioning tool like Github.

- You are able to use Android Studio to complete the tutorial videos and this lab assignment on either your own machine or the EECS remote labs.

Requirements of this Lab

- For this lab, you will be graded not only by JUnit tests given to you, **but also additional tests** covering some other input values. This is to encourage you to take more responsibility for the correctness of your code, by writing your own tests.
- **Programming Requirements**
  - You are **only allowed** to use primitive arrays (e.g., `int[]`, `String[]`, `Facility[]`) for declaring attributes and implementing methods.
  - Any use of a Java library class or method is forbidden (that is, use selections and loops to build your solution from scratch instead):
    - Here are some examples of **forbidden** classes/methods: `Arrays` class (e.g., `Arrays.copyOf`), `System` class (e.g., `System.arrayCopy`), `ArrayList` class, `String` class (e.g., `substring`).
    - The use of some library classes does not require an `import` statement, but these classes are **also forbidden** to be used.
    - Here are the exceptions (library methods which you **are allowed** to use if needed):
      - `String` class (`equals`, `format`)

    Violating this requirement will cause a **50% penalty** on your lab marks.
- Your lab submission will **only** be graded using JUnit (for this lab, the tests supplied to you plus some additional tests).
- For the JUnit test class `Tests.java` given to you:
  - Do **not** modify the test methods given to you.
  - You are allowed to add new nest methods.
- For each method which you are required to implement, derived from the JUnit test methods:
  - No `System.out.println` statements should appear in it.
  - No Scanner operations (e.g., `input.nextInt()`) should appear in it.

    Instead, declare input parameters of the method as indicated by the JUnit tests.
- You are welcome to ask questions related to this lab or the assigned tutorial videos on the forum. However, please be **cautious**:
  - You can help your fellow students understand the requirements of tasks.
  - **Do not share the code you developed to ask, or to answer, questions.**
    - Questions specific to the code you write would be best and most effectively addressed by TAs (during scheduled labs) or your instructor (during office hours or appointments).
    - Week 9's tutorial videos shows to you **debugger in Android Studio** (which should be used in the same manner as Eclipse debugger). You are encouraged to **set breakpoints and launch the debugger** when you are stuck at your own program.
  - **Hints** on how the solution should look like are **left only to the instructors** who moderate the forum.

1 Task 1: Complete Weekly Java Tutorial Videos

- For Lab7, you are assigned to study **Week 9 Part A1 to Part B4** of the Java tutorial series:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_64z4oGXy0kHBO7_gPkjNmG [Week 9 only]

  To reference tutorial videos from the previous weeks, see:

  https://www.youtube.com/playlist?list=PL5dxAmCmjv_6wy2m0yq2wObIWPz4tAxW6 [All Weeks]

  These Java tutorial videos assigned to you are meant for you to:

  1. Obtain extra hands-on programming experience on Java, supplementing your weekly lectures.
  2. Complete the lab assignment with the necessary skills and background.

  Though we do **not** require the submission of the weekly Java tutorial project (like in Lab0), **examples and insights discussed in these tutorials will be covered in your (written and programming) tests and exam**: should you decide to skip the weekly tutorial videos, it would be your choice.

  As you study through the example Java classes in the tutorial videos, you are advised to **type them out** (but absolutely feel free to add new Java classes to experiment) on Android Studio.

- You can find the iPad notes of illustrations from the tutorial videos here:

  https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/EECS1022%20Tutorial%20on%20Java.pdf

Notes from Lab6 and Lab7

- See this notes on how to manipulate objects with reference-typed, multi-valued attributes:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Tracing_PointCollectorTester.pdf

- See this notes on how to infer classes and methods from given JUnit tests:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Inferring_Classes_from_JUnit.pdf

- You can find here the example covered in the notes for practice:
  - Starter: https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Inferring_Classes_from_JUnit.zip
  - Solution: https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Inferring_Classes_from_JUnit_Solution.zip

2 Task 2: Setup an Android Studio Project

Starting Task 2 should mean that you have already completed the weekly Java tutorial videos (Section 1).

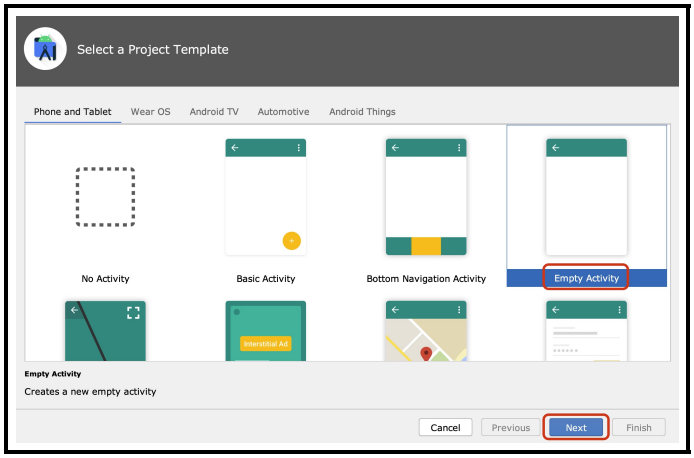
2.1 Step 1: Install Android Studio

See the detailed PDF instructions "**Install Android Studio 3.6.3 File**" on the M,N,O eClass site: https://eclass.yorku.ca/eclass/pluginfile.php/2004743/mod_resource/content/2/Installing_Android_Studio_v3.6.3.pdf.

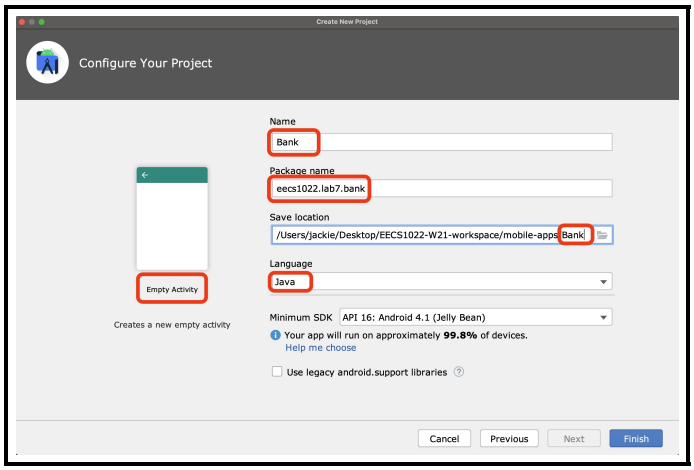
2.2 Step 2: Set up an Empty Activity in Android Studio

Unlike previous labs, there is no project archive file for you to import. Instead, you are asked to create an **Empty Activity** in Android Studio:

1. Launch Android Studio and click on `Create New Project`.
2. Under `Select a Project Template`, select `Empty Activity`.

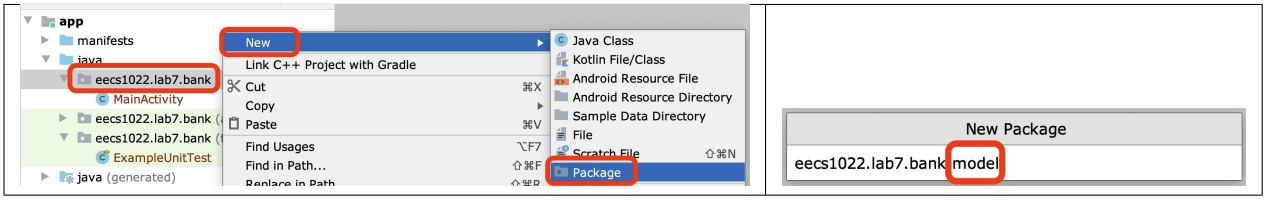


3. Under `Configure Your Project`:
   - Name: type **Bank**
   - Package Name: type **eecs1022.lab7.bank**
   - Save Location: make sure the last part of the path should match the app name **Bank**
   - Language: choose **Java**

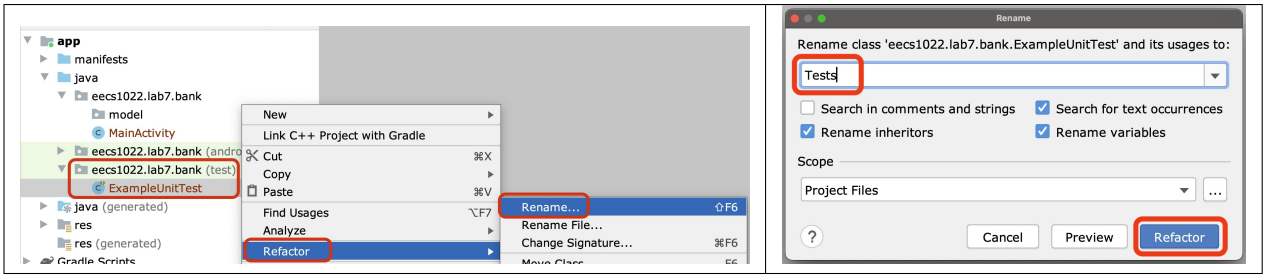


4. After the new project is successfully built:

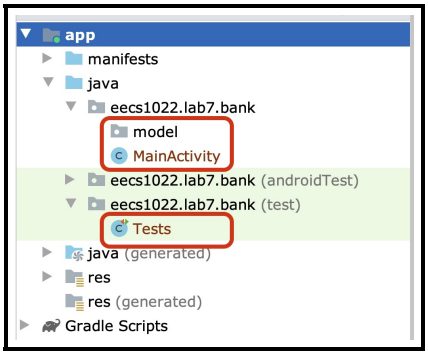
   4.1 Under `app/java/eecs1022.lab7.bank`, create a new package called `model`.



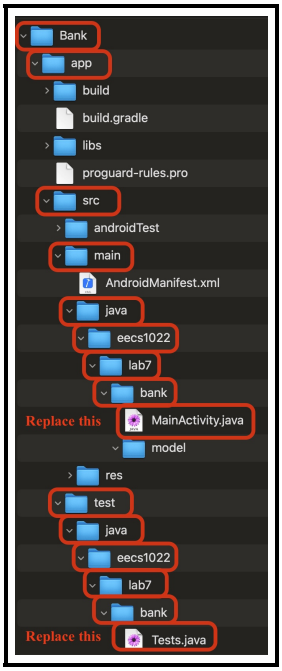
   4.2 Under `app/java/eecs1022.lab7.bank (test)`, rename the JUnit test class `ExampleUnitTest` to `Tests`.



   4.3 After performing the above steps, your project structure in Android Studio should look like:



2.3 Step 3: Download and Replace the Controller Class and JUnit Test Class



1. From eClass, download the archive file `EECS1022_W21_Lab7.zip`, containing:
   - The controller class: `MainActivity.java`
   - The JUnit test class: `Tests.java`
2. As shown in the left figure, do the following replacements:
   - Copy and replace the given `MainActivity.java` to your project folder: `Bank/app/src/main/java/eecs1022/lab7/bank`
   - Copy and replace the given `Tests.java` to your project folder: `Bank/app/src/test/java/eecs1022/lab7/bank`

     It is expected that **compilation errors** exist in both given files when first replaced into your project. See Section 3.2.1.

3 Task 3: Develop a Mobile App in Android Studio

3.1 Step 1: Study the Bank Problem

You are required to develop an object-oriented program solving the problem of a (simplified) bank, which provides services (i.e., deposit, withdraw, transfer, and print statement) to a collection of up to 6 clients (i.e., when a 7th client is attempted to be added to the bank, there should be an error message displayed). Each client can be characterized by their name, numerical balance (always displayed with two digits after the decimal point), and a history (i.e., collection) of up to 10 transactions (for this you can assume that the user will not complete more than 10 transactions for each client). Each client should be identified by their name. We may want to perform one of the following kinds of transactions:

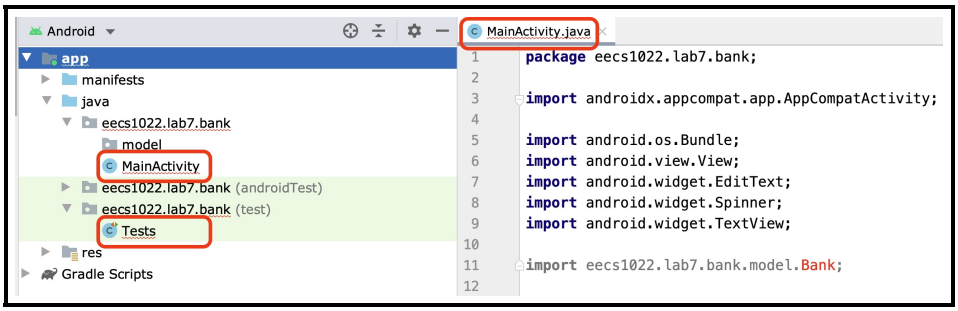
- Deposit some input `amount` of money **to** a client's account (identified by their name).
- Withdraw some input `amount` of money **from** a client's account (identified by their name).
- Transfer some input `amount` of money **from** a client's account **to** another client's account (identified by their names). In the case of transferring `$amount` from client $c1$ to client $c2$, it should be considered as two separate transactions: a withdrawal of `$amount` from client $c1$ and a deposit of `$amount` to client $c2$.

Also, throughout the life time of a bank object, given a client's name, we may want to it return a statement summarizing the client's status (i.e., account balance) and their history list of transactions.

3.2 Step 2: Using JUnit Tests to Develop Classes in Model Package

3.2.1 Inferring Model Classes from JUnit Tests

- It is **expected** that the `Tests` JUnit class (which you downloaded and copied from eClass) contains **compilation errors**. This is because that declarations and definitions of the required class(es) and method(s) it references are missing.



- The `model` package you created is empty. Class(es) and method(s) derived from the given JUnit class **must** be added to this package. You must **not** create any classes **not** indicated by the JUnit tests, otherwise your submitted model classes will **not** compile with the grading program.

  Therefore, your tasks are:

  1. Inferring from the given JUnit tests, add the missing class(es) and method(s) into the `model` package. Note a line from the `Tests` class (which specifies that all classes from the `model` package are referenced):

     ```
     import eecs1022.lab7.bank.model.*;
     ```

  2. Pass **all** JUnit tests given to you (i.e., a **green bar**).

  You must not modify these given JUnit tests, as they suggest how the intended class(es) and method(s) should be declared.

How to Deal with a Failed JUnit Test? Like the previous labs, in Android Studio, place breakpoints around assertions of the failed tests, launch the debugger, and then use the "step over", "step into", "step out", `Variables`, and `Watches` tools to help fix your code. See **Java Tutorials Week 9** for how to use all these.

3.2.2 Status of Bank

A bank's status reports whether or not the **last** invoked service (i.e., add a new account, deposit, withdraw, transfer, or print statement) results in an error:

- When there is an error upon invoking the service, the bank's status should be the corresponding error message.
- Otherwise, when there is no error upon invoking the service, the bank's status should be a summary of each stored account's status (see the given JUnit tests).

When multiple errors occur simultaneously, exactly one message for the lowest-ranked error is displayed. For example, say the number of clients has not reached the maximum yet, then given a clashing name (an error with rank 2) and a non-positive initial balance (an error with rank 3), we only output the error message `Error: Client` n `already exists`.

Add a New Account

Inputs: Name of client (say n) and the Initial Balance of client (say *bal*)

Rank	Error Condition	Error Message
1	Number of accounts in bank has reached its maximum	`Error: Maximum Number of Accounts Reached`
2	n clashes with the name of some existing client	`Error: Client` n `already exists`
3	*bal* is non-positive (i.e., ≤ 0)	`Error: Non-Positive Initial Balance`

Confirm: Case of a `DEPOSIT` Transaction

Inputs: `DEPOSIT` chosen as the service type, name of to-account (say *toName*), and amount of transaction (say *amount*).

Rank	Error Condition	Error Message
1	*toName* is not an existing client's name	`Error: To-Account` *toName* `does not exist`
2	*amount* is non-positive (i.e., ≤ 0)	`Error: Non-Positive Amount`

Confirm: Case of a `WITHDRAW` Transaction

Inputs: `WITHDRAW` chosen as the service type, name of from-account (say *fromName*), and amount of transaction (say *amount*).

Rank	Error Condition	Error Message
1	*fromName* is not an existing client's name	`Error: From-Account` *fromName* `does not exist`
2	*amount* is non-positive (i.e., ≤ 0)	`Error: Non-Positive Amount`
3	*amount* is larger than from-account's balance	`Error: Amount too large to withdraw`

Confirm: Case of a `TRANSFER` Transaction

Inputs: `TRANSFER` chosen as the service type, name of from-account (say *fromName*), name of to-account (say *toName*), and amount of transaction (say *amount*).

Rank	Error Condition	Error Message
1	*fromName* is not an existing client's name	`Error: From-Account` *fromName* `does not exist`
2	*toName* is not an existing client's name	`Error: To-Account` *toName* `does not exist`
3	*amount* is non-positive (i.e., ≤ 0)	`Error: Non-Positive Amount`
4	*amount* is larger than from-account's balance	`Error: Amount too large to transfer`

Confirm: Case of a `Print Statement` Service

Inputs: Name of a client (say n)

Error Condition	Error Message
n is not an existing client's name	`Error: From-Account` n `does not exist`

3.2.3 Hints and Requirements

- See this notes on how to declare and manipulate reference-typed, multi-valued attributes:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Tracing_PointCollectorTester.pdf

- See this notes on how to infer classes and methods from given JUnit tests:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_Inferring_Classes_and_Methods_from_JUnit_Tests.pdf

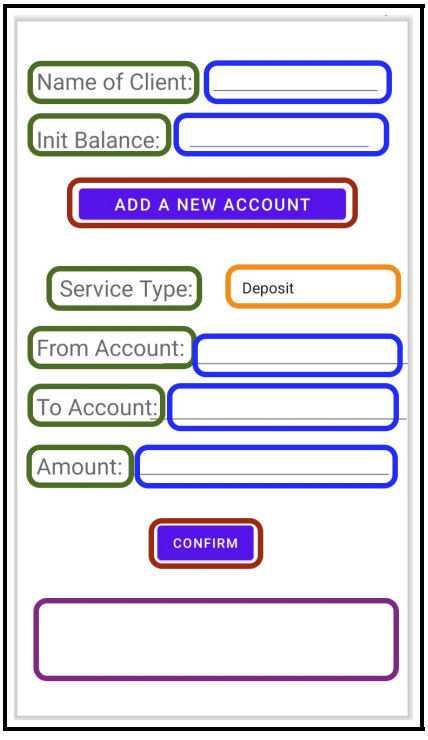
  Programming IDEs such as Eclipse and Android Studio are able to fix such compilation errors for you. **However, you are advised to follow the guidance as specified in the notes to fix these compilation errors manually, because: 1) it helps you better understand how the intended classes and methods work together; and 2) you may be tested in a written test or exam without the assistance of IDEs.**

- Any new class(es) you add must reside in the `model` package.
  - Once the necessary class(es) and method(s) are declared, you can add as many attributes as necessary to implement the body of each method.
  - Study carefully example instances as specified in `Tests.java`: they suggest the how the intended class(es) and method(s) should be declared and implemented.
  - Focus on *gradually* passing one test at a time.

3.3 Step 3: Design a GUI Layout

3.3.1 Given GUI Layout to Implement

You are then asked to implement the following GUI design:



In the left GUI design:

- Textviews (for displaying input prompts) are boxed in **green**.
- Plain Texts (or textfields for reading user-typed inputs) are boxed in **blue**.
- The spinner (for displaying a drop-down menu of options) is boxed in **orange**.
- Buttons (for invoking attached controller methods) are boxed in **red**.
- The (invisible) textview (for displaying computation results) is boxed in **purple**.

To implement the above GUI design, open `activity_main.xml` (under `app/src/main/res/layout`) in Android Studio.

Then, drag, drop, and organize the above specified GUI components. For each GUI component, be sure to set the appropriate `text`, as suggested above, and set its `id` to be used later in the controller (`MainActivity.java`).

3.3.2 Assumed Usage Pattern of the App

For the simplicity of this lab, you can assume that users (or testers) of your app behave in the following way:

1. Adding a New Account
   - 1.1 Enter the name of a new client.
   - 1.2 Enter the initial balance of this new client.
   - 1.3 Click on the button "ADD A NEW ACCOUNT"

     This step should cause your program to create a `Client` object, and add it to the `Bank` object's array attribute `clients`.

     See Section 3.2.2 for the possible error messages to be displayed.
2. Completing a New Transaction
   - 2.1 Select a service type (i.e., `DEPOSIT`, `WITHDRAW`, `TRANSFER`).
   - 2.2 Enter the To and/or From account owners:
     - In the case of `deposit`, input on the `To-Account` textfield should be retrieved; any input on the `From-Account` textfield should be ignored.
     - In the case of `withdraw`, input on the `From-Account` textfield should be retrieved; any input on the `To-Account` textfield should be ignored.
   - 2.3 Enter the amount for the selected service type.
   - 2.4 Click on the "CONFIRM" button.

     In cases of `DEPOSIT` and `WITHDRAW`, this step should cause your program to add a new transaction to, respectively, the to-account and the from-account. In the case of `TRANSFER`, a "withdraw" transaction should be added to the from-account, and a "deposit" transaction should be added to the to-account.

     See Section 3.2.2 for the possible error messages to be displayed.
3. Printing the Statement
   - 3.1 Select the service type (i.e., `Print Statement`).
   - 3.2 Input on the `From-Account` textfield should be retrieved; any input on the `To-Account` textfield should be ignored.
   - 3.3 Click on the "CONFIRM" button.

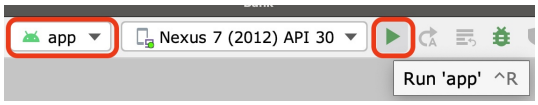
     See Section 3.2.2 for the possible error messages to be displayed.

The two buttons "ADD A NEW ACCOUNT" and "CONFIRM" may be clicked in any order.

3.4 Step 4: Develop the Controller Class

From Section 2.3, you already copied and replaced the given controller class `MainActivity`.

- Follow through the comments/hints in the given controller file to complete it.
- Refer to this short demo tutorial video on how your Android mobile app is supposed to work: https://www.youtube.com/watch?v=xal9NkIs46U&list=PL5dxAmCmjv_7YgI2LgcwjWTHiNZSR-aQX&index=2
- Test the app by launching it using an emulator in Android Studio:



- The app name on the top of the emulator screen must read:

  `Bank App - FirstName LastName (StudentNumber)`

  where your first name, last name, and student number are displayed in the suggested order.

  Hint: Go to the resource file `app/res/values/strings.xml` and modify the `app_name` attribute accordingly.

4 Submission

1. Before you submit, you must make sure that there are **no compilation errors** in the given JUnit test class and in all classes you have created in the `model` package.

   Submitting programs with errors (meaning that it cannot be run for grading) will result in possible partial, but low, marks.

2. Go to the eClass site for Sections M,N,O: https://eclass.yorku.ca/eclass/course/view.php?id=6214

3. Under the `Lab Submissions` section, click on `Lab7` to submit the following list of files:

   - **All** Java files in the `model` package (as indicated by the given `Tests.java`):
     `Bank/app/src/main/java/eecs1022/lab7/bank/model`
   - The completed controller file: `Bank/app/src/main/java/eecs1022/lab7/bank/MainActivity.java`
   - The completed GUI layout file: `Bank/app/src/main/res/layout/activity_main.xml`
   - The strings resource file: `Bank/app/src/main/res/values/strings.xml`

   Note:

   - You may **upload** as many draft versions as you like before the deadline.
   - You must explicitly **submit** the draft version for grading before the deadline.
   - **Once you click on the `submit` button, you can no longer upload another draft version.**