|  |
| --- |
| **Can I reserve a table?** |

**Resources and References**

CheckBox - <http://developer.android.com/guide/topics/ui/controls/checkbox.html>

DatePicker & TimePicker - <http://developer.android.com/guide/topics/ui/controls/pickers.html>

Toast - <http://developer.android.com/guide/topics/ui/notifiers/toasts.html>

Debugging - <https://developer.android.com/studio/debug/index.html>

Table of Contents

[Understanding TimePicker and DatePicker 2](#_Toc71414799)

[Create an Android Project 2](#_Toc71414800)

[Section A: Design the Layout 3](#_Toc71414801)

[Section B: Understanding Date and Time Libraries 6](#_Toc71414802)

[Introducing CheckBox 10](#_Toc71414803)

[Create a New Project for Checkbox Demo 10](#_Toc71414804)

[Section D: Design the Layout 11](#_Toc71414805)

[Section E: Handle the User Events 11](#_Toc71414806)

[A Toast! 12](#_Toc71414807)

[Introducing Logcat Messages 13](#_Toc71414808)

[Introducing Debugging 15](#_Toc71414809)

[Back to the Problem Statement 22](#_Toc71414810)

# Understanding TimePicker and DatePicker

Recall in the last 2 problems that we were solving programming problems in creating layout design and eventually completing an app with simple business logic. In today’s worksheet, you will be learning new UI elements, and code the necessary business logics inside the onCreate() method of the Activity class.

Do also use this worksheet as another opportunity to strengthen your skills in using the Android Studio IDE, as well as practicing different layout implementation.

## Create an Android Project

We will create an app to demonstrate the use of TimePicker and DatePicker. The app has a TimePicker, a DatePicker, two Buttons and a TextView. The user can modify the time and date through the TimePicker and DatePicker and display the time and date in the TextView by clicking on the respective Buttons.

1. In Android Studio, create a new project with the following information.

|  |  |
| --- | --- |
| **Project Template** | Empty Activity |
| **Application Name** | Demo Time and Date App |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.demotimeanddateapp |
| **Project Location** | D:\C346\Workspace\DemoTimeandDateApp |
| **Language** | Java |
| **Minimum API Level** | API 16 |

Update activity\_main.xml to use the following template.

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:orientation="vertical"**>  </**LinearLayout**> |

Basic LinearLayout Template

**Note:** We will be using LinearLayout throughout this module. Please use LinearLayout when answering questions and quizzes.

## Section A: Design the Layout

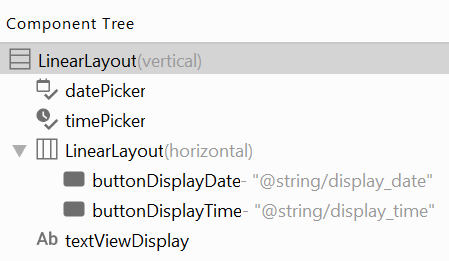
1. Add in a DatePicker followed by a TimePicker.

|  |
| --- |
| <**DatePicker  android:id="@+id/datePicker"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"** />  <**TimePicker  android:id="@+id/timePicker"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"** /> |

1. As illustrated in Figure 1a, you will realize that there is no space left in the device screen to add in other UI elements as shown in Figure 1b.

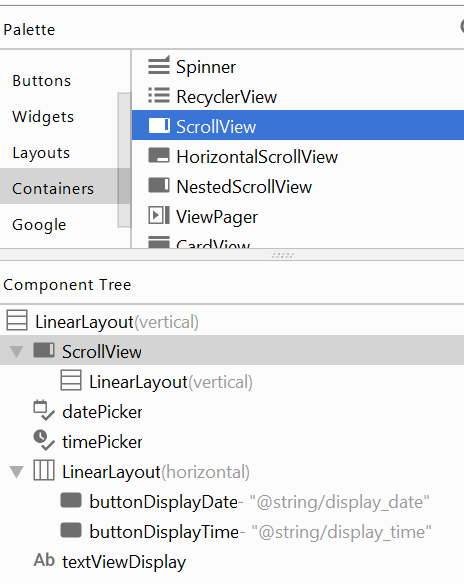
|  |  |
| --- | --- |
|  | C:\Users\denise_quek\Desktop\Screenshot_1589340189.png  No more space at the bottom |

1. Continue to finish up the design as given below. The texts of the buttons are "Display Date" and "Display Time" respectively.

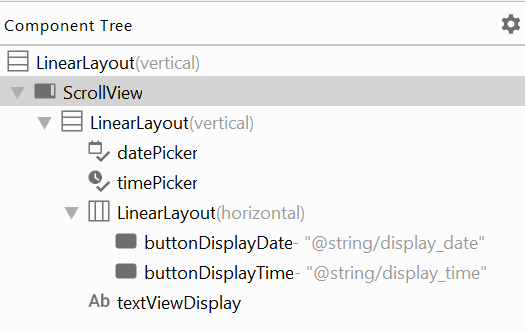


Note that the 2 buttons and the text views are not visible in the viewport of the device screen. Also, "textViewDisplay" is not inside the nested layout.

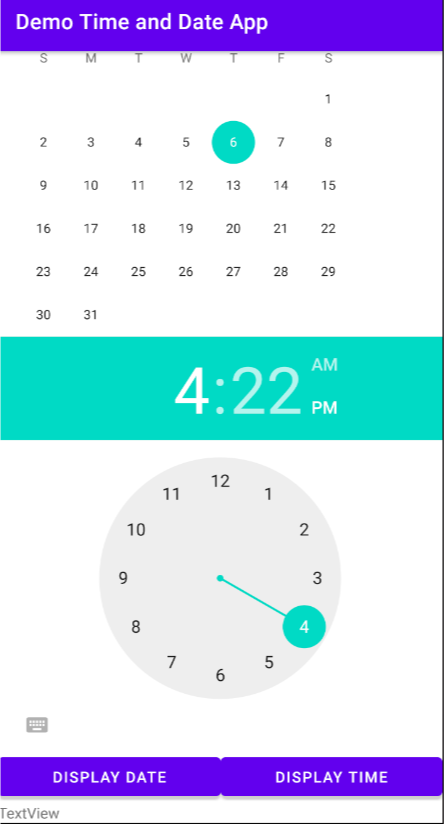
1. There is a solution to make the viewport scrollable when in emulator or on actual device, but not in the UI design. To enable scrolling, we need to add a **ScrollView** to the layout and put all the UI elements inside except the top-level LinearLayout.
   1. Find **ScrollView** in Containers, and drag and drop it on LinearLayout (vertical), you should see it as below.



* 1. Drag each of the UI elements to be put inside the ScrollView and drop it on the LinearLayout (vertical) in the ScrollView. Your final Component Tree should be the same as the screenshot given below.



1. Run the app on the emulator. You should be able to see all the UI elements by scrolling up and down the screen.

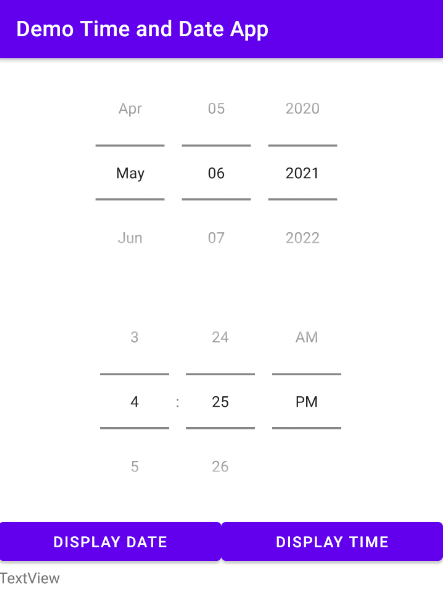


1. For ease of entry, you could modify the DatePicker design by adding the following attributes to the DatePicker UI element.

Add in the following two attributes to the DatePicker element.

|  |
| --- |
| **android:datePickerMode="spinner" android:calendarViewShown="false"** |

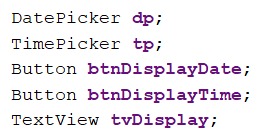
1. You may add the similar attribute "timePickerMode" to the TimePicker. Test the app on the emulator again. This time you should be able to modify the date and time easily.



## Section B: Understanding Date and Time Libraries

Let’s learn how to implement the business logic by writing the necessary code to display the date and time selection onto the TextView.

1. Follow the code below to declare the variables in the MainActivity.



1. Bind (Link) the variables to their respective UI elements in onCreate() method.

|  |
| --- |
| **dp**=findViewById(R.id.***datePicker***); **tp**=findViewById(R.id.***timePicker***); **btnDisplayDate**=findViewById(R.id.***buttonDisplayDate***); **btnDisplayTime**=findViewById(R.id.***buttonDisplayTime***); **tvDisplay**=findViewById(R.id.***textViewDisplay***); |

1. When the Button “Display Time” is clicked, we want to display the time of the TimePicker in the TextView. We have learnt from last lesson on:
   1. Handle User Event, like button click
   2. Display text in TextView

However, we haven’t learnt how to extract the time value from the TimePicker UI element. Take a look at the following reference:- <https://developer.android.com/reference/android/widget/TimePicker>

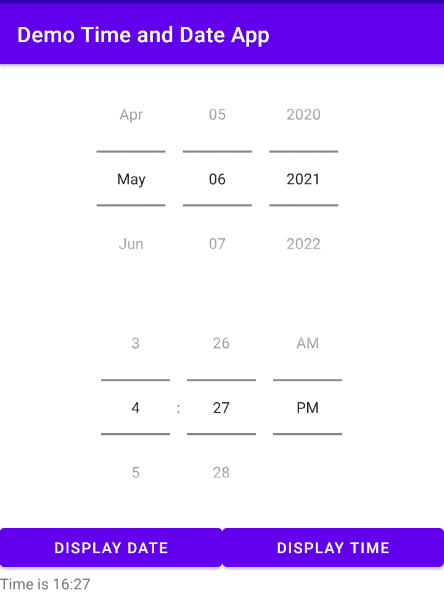
Find out the methods to get the time and set the time. Fill in the table below.

|  |  |  |
| --- | --- | --- |
| **Method**  **(Below API level 23)** | **Method**  **(API level 23 and above)** | **Definition** |
| getCurrentHour() | getHour() | Returns the currently selected hour using 24-hour time. |
| getCurrentMinute() | getMinute() | Returns the currently selected minute. |
| setCurrentHour(int hour) | setHour(int) | Sets the currently selected hour using 24-hour time. |
| setCurrentMinute(int currentMinute) | setMinute(int) | Sets the currently selected minute. |

|  |  |
| --- | --- |
| ? | Currently your minimum SDK is API 16. Which method should you use to return the selected hour from a TimePicker?  What if you set your minimum SDK to API 24? |
| API 16: getCurrentHour().  API 24 : getHour(). | |

1. Write code to handle the "Display Time" button click event. The text should show "Time is H:m", e.g., "Time is 16:35". Copy and paste your code in the box below.

|  |
| --- |
| **btnDisplayTime**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  String output=**"Time is "** + String.*valueOf*(**tp**.getCurrentHour()) + **":"** + String.*valueOf*(**tp**.getCurrentMinute());  **tvDisplay**.setText(output);  } }); |



1. Now let’s move on to handle the event when user clicks the "Display Date" button. Take a look at the following reference:- <https://developer.android.com/reference/android/widget/DatePicker>

Find out the methods to get the date and set the date. Fill in the table below.

|  |  |
| --- | --- |
| **Method** | **Definition** |
| [getYear](https://developer.android.com/reference/android/widget/DatePicker.html#getYear())() | Returns the selected year. |
| getMonth() | Returns the selected month. |
| getDayOfMonth() | Returns the selected day of month. |
| updateDate(int year, int month, int dayOfMonth) | Update the current date. |

1. Write code to handle the "Display Date" button click event. The text should show "Date is D/M/Y", e.g., "Date is 29/11/2017". Copy and paste your code in the box below.

|  |
| --- |
| **btnDisplayDate**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  String date=**"Date is "** + String.*valueOf*(**dp**.getDayOfMonth()) + **"/"** + String.*valueOf*(**dp**.getMonth()) + **"/"** + String.*valueOf*(**dp**.getYear());  **tvDisplay**.setText(date);    } }); |

1. Test the code and verify that the TextView shows the same date as the DatePicker after clicking the button "Display Date".

|  |  |
| --- | --- |
| ? | Does the TextView show the date correctly for you? If not, identify the error and explain it (Google is your friend). |
| No, because the getMonth() method starts from 0. | |

1. Correct your code to fix the error.

|  |
| --- |
| **btnDisplayDate**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  String date=**"Date is "** + String.*valueOf*(**dp**.getDayOfMonth()) + **"/"** + String.*valueOf*(**dp**.getMonth()) + **"/"** + String.*valueOf*(**dp**.getYear());  **tvDisplay**.setText(date);  } }); |

1. To enhance the app, we would like to have a Reset button to set the TimePicker to 12AM and the DatePicker to Jan 01, 2020. Follow the steps below to implement it.
   1. Add in the UI element (Button) to the UI layout. You can add it at the bottom.
   2. Declare the variable btnReset in MainActivity.java.
   3. Link it to the UI element in onCreate().
   4. Handle the button click event. Refer to the tables above on the methods to set the date and time

Test your app and make sure it works as expected. Copy and paste your code for part d below.

|  |
| --- |
| **btnReset**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View view) {  **tp**.setCurrentHour(0);  **tp**.setCurrentMinute(0);  **dp**.updateDate(2020,0,1);  } }); |

|  |  |
| --- | --- |
|  | **Learning Checkpoint 1** |
| *By now, you should be able to*   * *Draw UI via the IDE designer tool in Design mode or Text mode with ease.* * *Bind (Link) the class variables to their respective UI elements.* * *Write code to perform event handling.*   **To recap on what we have learnt so far,**  Implementing the layout is just the starting point of creating your Android program, and the program would be non-functional without the business logics, such as your code to perform event handling, including code to handle button click action.  Next, as a programmer, do expand your knowledge on the repertoire (range) of libraries that you could apply in your code, like what we have explored earlier about the DataPicker and TimePicker classes. | |

# Introducing CheckBox

Let’s learn about a new UI element, Checkbox.

## Create a New Project for Checkbox Demo

We will create an app to demonstrate the use of Checkbox. The app has a Checkbox, a Button and a TextView. The user can check or uncheck the Checkbox. Once the Button is clicked, the TextView will show the state of the Checkbox.

Create a new Android project with the following details:

|  |  |
| --- | --- |
| **Project Template** | Empty Activity |
| **Application Name** | Demo Checkbox Example |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.democheckboxexample |
| **Project Location** | D:\C346\Workspace\DemoCheckboxExample |
| **Language** | Java |
| **Minimum API Level** | API 16 |

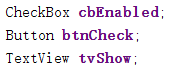
## Section D: Design the Layout

Design the UI layout (activity\_main.xml) according to the Component Tree and device screen view given below.

|  |  |
| --- | --- |
|  |  |

## Section E: Handle the User Events

1. We need to declare the variables and link them to the UI elements first. Declare the following variables in the MainActivity.



1. Bind the variables to their respective UI elements in onCreate() method.
2. The methods below are to get and set the state for checkbox. Search online and find out how to use them.

|  |  |
| --- | --- |
| **Method** | **Definition** |
| [isChecked](https://developer.android.com/reference/android/widget/CompoundButton.html#isChecked())() | Returns the state of this checkbox. |
| [setChecked](https://developer.android.com/reference/android/widget/CompoundButton.html#setChecked(boolean))(boolean checked) | Changes the checked state of this checkbox. |

1. Write code to handle the button click event. The text should show “The discount is given/not given” depending on the state of the checkbox. The device screen should yield the following sample output:

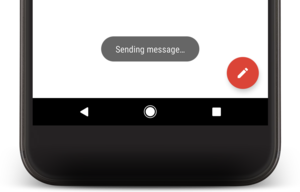
|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| **btnCheck**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  **if**(**cbDiscount**.isChecked())  {  **tvDisplay**.setText(**"The discount is given."** );  }  **else** {   **tvDisplay**.setText(**"The discount is not given."**);  }  } }); |

# A Toast!

A Toast in Android works like a notification message to users, and it fades after displaying a predefined message for a short or long moment.

1. If you use Android phone, you should have seen Toasts before. A typical example is given below.



1. Search "toasts android" online and write down the purpose of using Toasts.

|  |
| --- |
| The purpose is displaying a small message and it only remains visible for a short time period. |

The following methods are used to create a toast message.

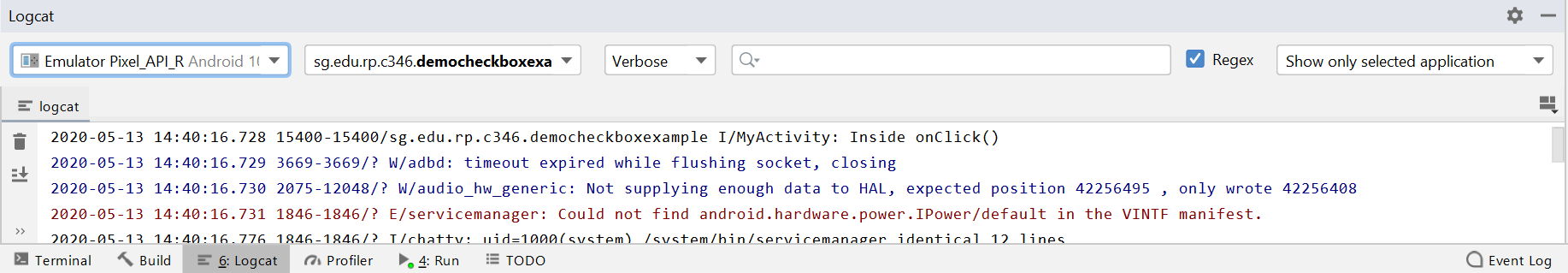
|  |  |
| --- | --- |
| **Method** | **Definition** |
| [makeText](https://developer.android.com/reference/android/widget/Toast.html#makeText(android.content.Context,%20java.lang.CharSequence,%20int))([Context](https://developer.android.com/reference/android/content/Context.html) context, [CharSequence](https://developer.android.com/reference/java/lang/CharSequence.html) text, int duration) | Make a standard toast that just contains a text view. |
| [show](https://developer.android.com/reference/android/widget/Toast.html#show())() | Show the view for the specified duration. |

1. Modify your code to add a toast message to show “Button Click” for a long duration when the “Check” Button is clicked. Test your code and make sure it works as expected. Copy and paste your updated code handling the button click event below.

|  |
| --- |
| **btnCheck**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {    **if**(**cbDiscount**.isChecked())  {  **tvDisplay**.setText(**"The discount is given. You need to pay "** + pay);  }  **else**  {  **tvDisplay**.setText(**"The discount is not given. You need to pay "** +pay);  }   Toast.*makeText*(MainActivity.**this**,**"Button click"**,Toast.***LENGTH\_LONG***).show();    } }); |

# Introducing Logcat Messages

Logcat is a great library and tool to monitor the status of your program execution. While the app is running, logcat displays a lot of system messages. We can make use of it to log some information or state. You can open the log window by clicking on the "Logcat" button at the bottom of your Android Studio.



1. Study the following link to find out how to create different types of log in your code.

<http://developer.android.com/tools/debugging/debugging-log.html#logClass>

|  |  |
| --- | --- |
| **Log Type** | **Method** |
| verbose | Log.v(String, String) |
| debug | Log.d(String, String) |
| information | Log.i(String, String) |
| warning | Log.w(String, String) |
| error | Log.e(String, String) |

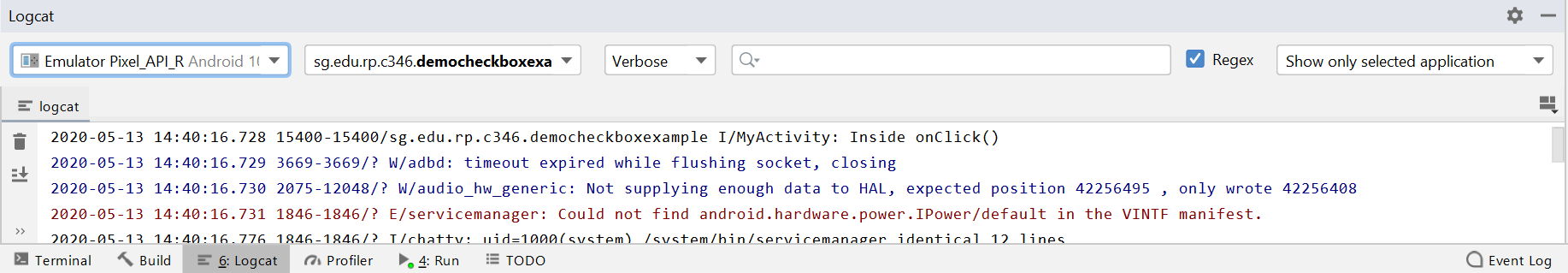
1. Explain the meaning of each parameter in the method:

|  |  |  |
| --- | --- | --- |
|  | **Data type** | **Meaning** |
| **Parameter 1** | String | Source |
| **Parameter 2** | String | method |

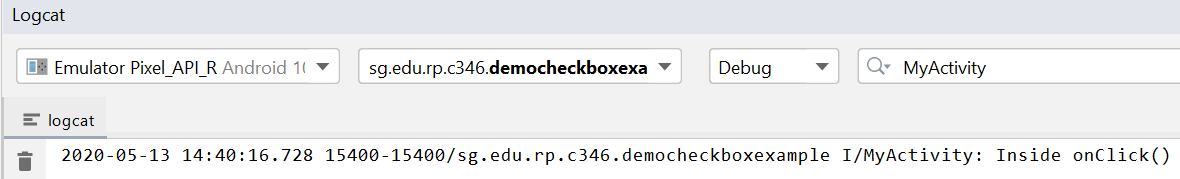
1. Try to insert the following code in the onClick() method as the first line of your updated code, and run your app.

|  |
| --- |
| Log.i("MyActivity", "Inside onClick()"); |

1. Click on the “Check” button in the app and look into the Logcat window in the Android Studio. You should be able to see the line of message in the red box below.



If there are too many messages to find a certain one, you can filter by filling the source, “MyActivity” in this case, in the filter box or selecting the log type from the dropdown list.

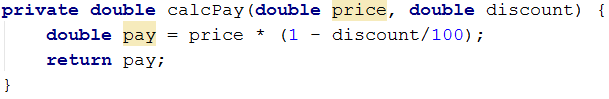


# Introducing Debugging

Debugging is a necessary skill that every developer should be equipped with. Android Studio provides a debugger for us to monitor our application in real time and track down bugs easily. Let’s learn how to perform a basic debugging in this section.

1. Create a method called “calcPay()” in MainActivity.java to calculate the pay based on the price and the discount. The logic is as the screenshot below.

Note: the method “calcPay()” should be outside the “onCreate()” method. We have learnt about Java method in C208.



1. Modify the "onClick()" method to calculate the pay by calling the "calcPay()" method and to display the pay on the TextView.

Assume the price is 100 and the discount is 20% if there is a discount (when the Checkbox is checked). The logic is as below.

|  |
| --- |
| **if**(**cbEnabled**.isChecked()) {  **double** pay = calcPay(100, 20);  **tvShow**.setText(**"The discount is given. You need to pay "** + pay); } **else** {  **double** pay = calcPay(100, 0);  **tvShow**.setText(**"The discount is not given. You need to pay "** + pay); } |

1. Now let’s debug the app to monitor how it runs as well as to familiarize us with the debugger. Set a breakpoint in the app code.

*A line breakpoint, the most common type of breakpoint, is to pause the execution of your app at a specified line of code. While paused, you can examine variables, evaluate expressions, then continue execution line by line to determine the causes of runtime errors.*

Say if we want to see how the program runs when we click on the “Check” button, we can set a breakpoint at line 30 as shown below, by clicking the left gutter along that line of code. A red dot should appear next to the line when you set a breakpoint.

|  |  |  |
| --- | --- | --- |
|  | | |
| ? | Why should we set the breakpoint at **line 30** if we want to see how the program runs when clicking on the “Check” button?  *Note: the line number (30) could be different from yours.* |
| Breakpoint will not work in our app, it will only work in the debug mode. What happen after clicking the check button. | |

1. Click the **Stop** button in the toolbar to stop running your app on Emulator if it is running.

|  |
| --- |
|  |

1. In the toolbar, click **Debug** button to display the **Select Deployment Target** window.

|  |
| --- |
|  |

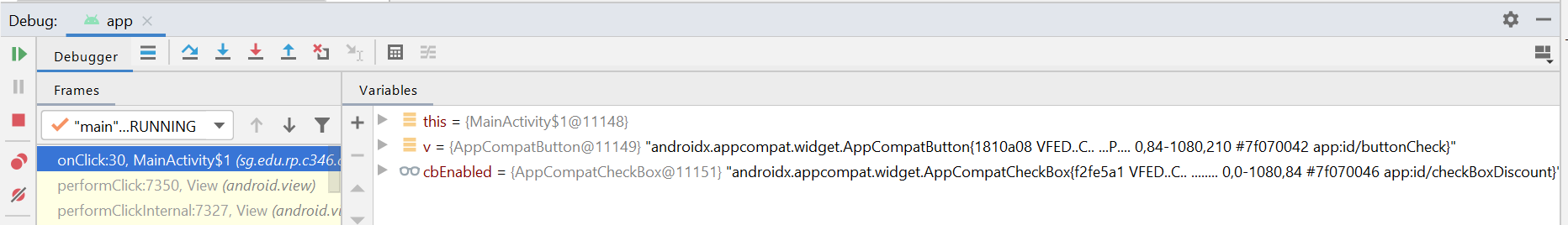
1. Android Studio will start building and installation. Once it has done, switch to your Emulator and click the “Check” button in your app.
2. Switch back to the Android Studio. What happen to the code execution after you click the “Check” button?

You may have noticed that, Android Studio pauses the execution of your app when it reaches the line where we set the breakpoint as below.

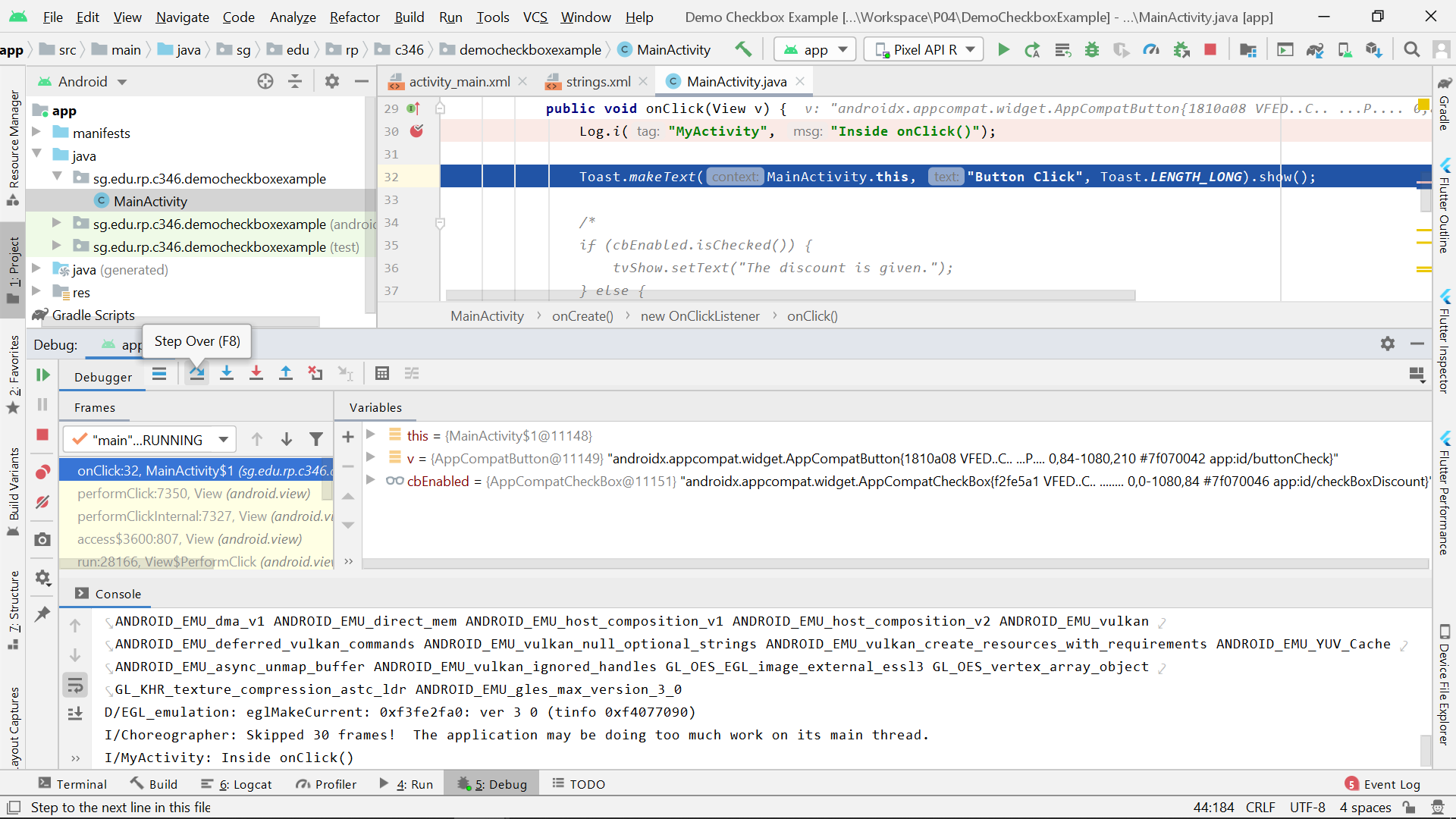
|  |
| --- |
|  |

1. Also, a **Debug** window is open at the bottom. You can use the tools in the **Debugger** tab to identify the state of the app now.

*Note: if the* ***Debug*** *window is not open, you can select* ***View > Tool Windows > Debug*** *to open it.*



1. Now let’s continue the execution line by line. Press the “**Step Over**” button in the **Debug** window.



You could have observed that, the execution advances to the next line once you click the “Step Over” button.

|  |
| --- |
|  |

This means that the previous line of code (line 30 in the screenshot above) has been executed. To verify that, go to the **Logcat** window, and find the log printed from line 30. Take a screenshot and paste it in the box below.

|  |
| --- |
|  |

1. Next, switch back to the **Debug** window, and press the “**Step Over**” button again to make the execution advance to the “if condition” at line 32 in the screenshot below.

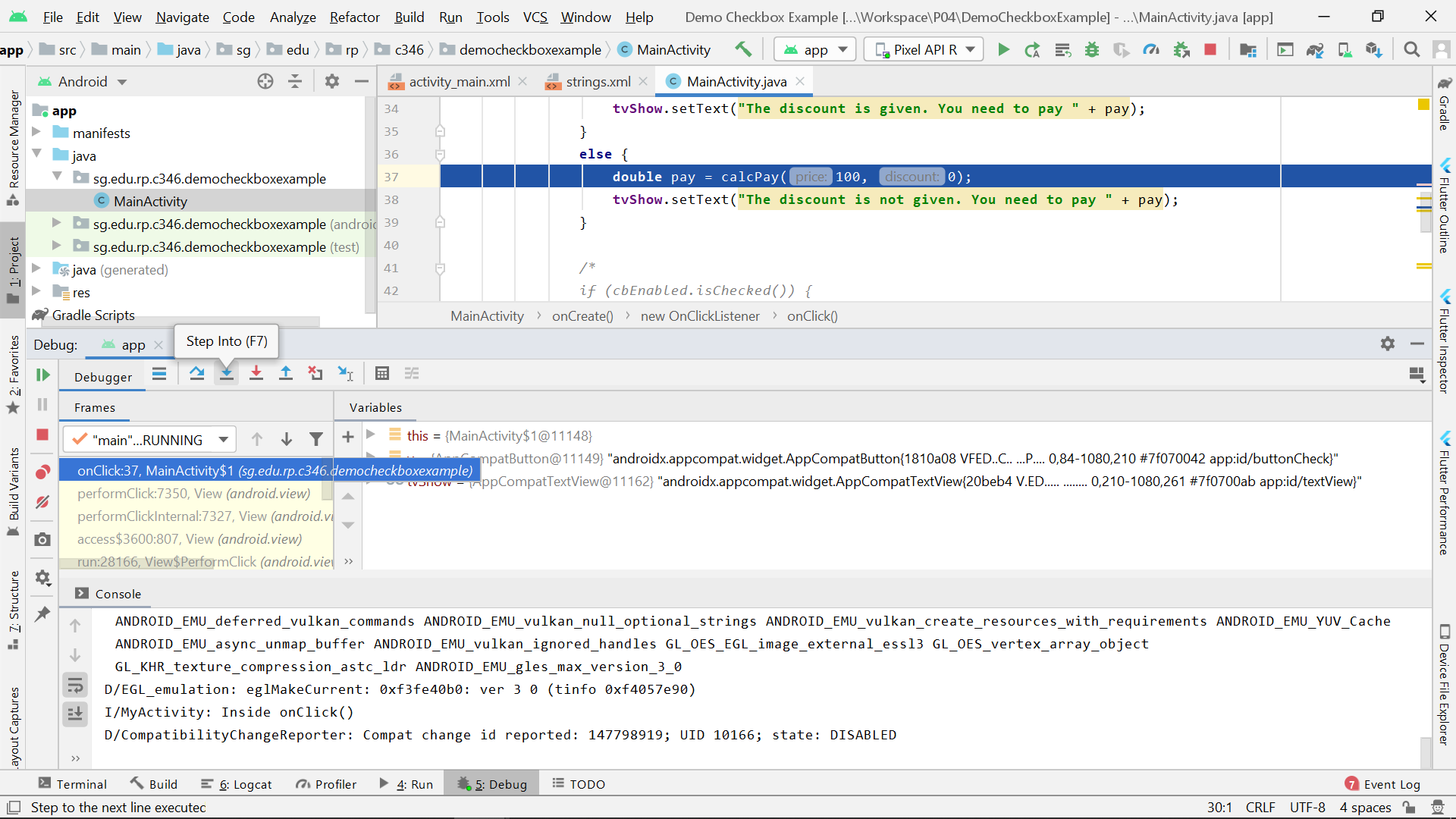
|  |
| --- |
|  |

|  |  |
| --- | --- |
| ? | If we click “**Step Over**” button again, which line of code in the screenshot above will the execution advance to? Why? |
| Line 33, as step over will move on to the next line. | |

1. Press the “**Step Over**” button again to verify your answer above.
2. Now the execution reaches a line where we will be calling the “calcPay()” method we created earlier. If we press “**Step Over**” button at this moment, the execution will advance to the next line without entering the method.

But sometimes, we may want to check the logic of the method as well, so we need to go in the method body before moving to the next line. To do this, we may use “**Step Into**”.

Click the “**Step Into**” button in the toolbar.



You should be able to see that, the execution advances to the method definition at line 44 in the screenshot below.

|  |
| --- |
|  |

1. Press “**Step Over**” button to continue the execution in the method body. Let the execution advance to the “return statement” at line 46 in the screenshot below.

During debugging, we may inspect variables. For example, we can observe the change of certain variable’s value, which may help us resolve an issue. We can do this by setting some watchpoint. But now let’s learn some easy ways to observe a variable’s value first.

There are at least three ways to observe a variable’s value as below.

1. Usually the value is printed at the end of each line in the editor.
2. Mouse over the variable to observe the value.
3. Observe the value in the **Variables** window at the bottom.

|  |
| --- |
|  |

1. If we continue “**Step Over**”, after line 47 (the end of the method) in the screenshot above, the execution will go back to the line where we were before coming in the method. Alternatively, if we have finished checking the method, we can use “**Step Out**” anytime to go back.

Press the “**Step Out**” button in the toolbar.

|  |
| --- |
|  |

The execution should advance to the line where we were before coming in the method.

1. To continue running the app normally, we may use “**Resume Program**”. If there is no other breakpoint in the following code, the execution will not be paused again.

Press the “**Resume Program**” button.

|  |
| --- |
|  |

Since there is no other breakpoint in our code, the app should resume running normally rather than be paused, after “**Resume Program**” button has been clicked.

|  |  |
| --- | --- |
| ? | Observe the Emulator, what text is displayed in the TextView? Explain why. |
| The discount is not given. You need to pay 100. | |

1. Now, check the Checkbox and click the “Check” button to perform a similar debugging for the scenario when the discount is given.

|  |  |
| --- | --- |
|  | **Learning Checkpoint 2** |
| *By now, you should be able to*   * *Use Toast library to implement Toast message.* * *Use Logcat library to implement logs that can track the run time program flow.* * *Use Debugger to track the run time program flow and inspect variable for troubleshooting.*   **To recap on what we have learnt so far,**  A toast provides simple feedback about an operation in a small popup. It does not interrupt the running application. It is useful in debugging by displaying the values of the variables and checkpoints in the program, although developer could have utilised the debugging features in Android Studio to uncover and correct runtime errors.  Logcat can be used to display system messages when the app is executed. It is useful for developer to analyse the program flow from the log information, especially when there is a bug in the Android program.  Debugging is a necessary skill for developer to monitor the application in real time and analyse the program flow. The execution will be paused at the breakpoint and the variables can be inspected easily. It also makes bugs easier to be tracked down and addressed.  Now, which is the best approach to uncover the bugs in your program during runtime?  Would it be  1) Using Toast messages?  2) Planting lots of logcat codes all over your program?  3) Using the built-in Debugger? | |

# Back to the Problem Statement

**Create a New Project for the Reservation App**

Create a new project for the problem statement based on the following details:

|  |  |
| --- | --- |
| **Project Template** | Empty Activity |
| **Application Name** | L04-Reservation |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.l04\_reservation |
| **Project Location** | D:\C346\Workspace\L04Reservation |
| **Language** | Java |
| **Minimum API Level** | API 16 |

**Design the layout**

1. Based on the Problem Statement, do a quick sketch on paper on how you would like the User Interface (UI) of the application to look like. Fill in the table below with the UI element and the number you may want to use for the layout:

|  |  |  |
| --- | --- | --- |
| **UI Element** | **Number of UI element needed** | **Purpose** |
| Button | 2 | 1 for making the reservation; 1 for resetting the EditTexts, TimePicker, DatePicker and CheckBox. |
| Plain Text | 1 | For entering the user’s name |
| Phone | 1 | For entering the user’s phone number |
| Number text | 1 | For entering the size of the group. |
| Date picker | 1 | For choosing a date |
| Time picker | 1 | For choosing a timing |
| Radio group and radio button | 1 radio group, 2 radio buttons | For selecting table in smoking or non-smoking area |
|  |  |  |

1. Design the UI layout for the application in Android Studio. Take a screenshot of final design you come out with and paste it in the box below.

*Note: Remember to change the layout to* ***LinearLayout (vertical)****.*

Before you move on to the Java coding, remember to

* assign proper IDs to the UI elements
* use string resources to display all the texts

**Your screenshot:**

|  |
| --- |
|  |

**activity\_main.xml**

|  |
| --- |
|  |

**strings.xml**

|  |
| --- |
|  |

**Handle the User Events**

Continue to finish up the rest of the code.

**GitHub repository URL**

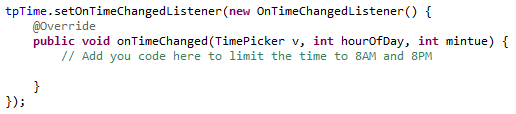
|  |
| --- |
|  |

**More challenges**

1. Check for empty fields: if any EditText field is empty, the reservation should not be made and an error message should be displayed using Toast.
2. There may be times when your app needs to be notified and take action when the user picks the time/date.

Scenario 1: Restrict the reservation time to only between 8AM and 8:59PM inclusive.

1. To do that, you need to setup the “OnTimeChangedListener” to the time change event for the TimePicker, which is similar to how you setup the “OnClickListener” for the button.
2. Like the button “OnClickListener”, you need to add the code below for setting the “OnTimeChangedListener” in the “onCreate()” method in the MainActivity class.



1. Like the “onClick()” method in the button “OnClickListener”, you need to implement the logic for the “onTimeChanged()” method in the “OnTimeChangedListener”.

How could the time roll back to 8PM if the user chooses 9PM or later?

*Hint: check the value of the method’s parameter, “hourOfDay”, and use the “setCurrentHour()” method as needed.*

How to check the value of the method’s parameter?

*Hint: Toasts, Logcat or Debugging.*

Scenario 2: Restrict the reservation to a date and time that is after today.