

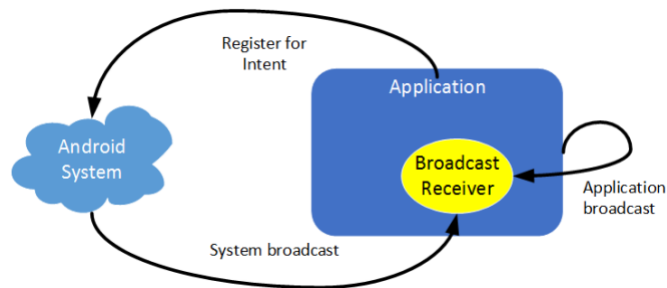
CMPS 312 Mobile Application Development

LAB 9: BROADCAST RECEIVERS AND NOTIFICATIONS

OBJECTIVE

In this lab you will learn how to use android BroadcastReceiver and Notification components.

- A **broadcast receiver** (short receiver) is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens.



- Android **notification** is a message you can display to the user outside of your application's normal UI.



By the end of the lab you should know, how to

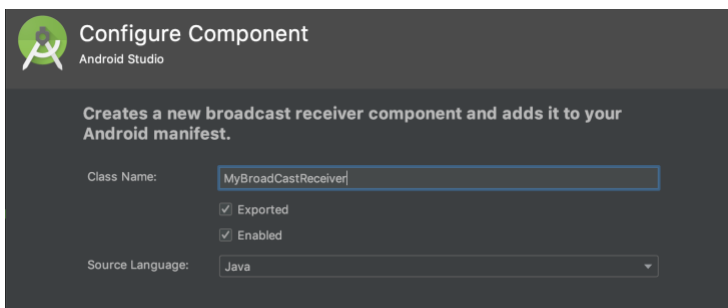
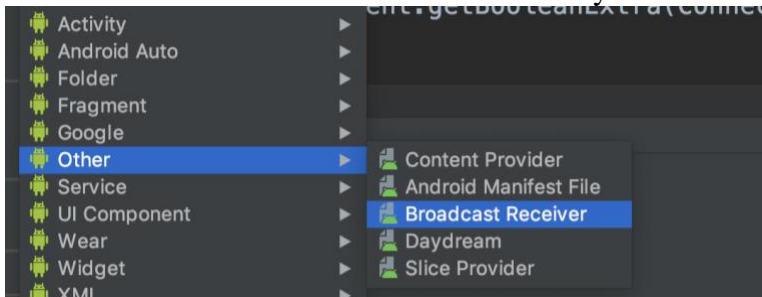
1. Create static broadcast receivers through the manifest
2. Register and unregister dynamic broadcast receivers
3. Send broadcast system wide
4. Build/Notify notifications
5. Use pending intents and much more

TUTORIAL

HOW TO CREAT BROADCAST RECEIVERS

We will create a simple app that listens to network connectivity of the device. When we have a network display connected otherwise we will show disconnected on the screen.

1. Create an application and name it “**SimpleBroadCastReciever**”
2. Create a broadcast receiver class and call it MyBroadcastReceiver



3. Remove the all the stuff inside the **onReceive** Method and display a Toast Message that says “network status changed”
4. Open your **MainActivity** and register the receiver
5. In the **onStart()** Method register your broadcast receiver

```
@Override
protected void onStart() {
    super.onStart();

    IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
    filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
    registerReceiver(receiver, filter);
    notificationManagerCompat = NotificationManagerCompat.from(this);
}
```

6. In the **onStop()** method unregister your broadcast receiver

```
this.unregisterReceiver(receiver);
```

7. Open your broadcast receiver and make it communicate to the MainActivity through interface to show the message [connected / disconnected] on the textview in the MainActivity.

```
public interface MyBroadcastInterface {
    void showMessage(String message);
}
```

8. The broadcast receiver should look something like this once you complete checking the connection of the device

```
MyBroadcastInterface anInterface;

@Override
public void onReceive(Context context, Intent intent) {
    anInterface = (MyBroadcastInterface) context;

    if (ConnectivityManager.CONNECTIVITY_ACTION.equals(intent.getAction())) {
        //it could be connected or disconnected
        boolean disconnected = intent
            .getBooleanExtra(ConnectivityManager.EXTRA_NO_CONNECTIVITY,
                defaultValue: false);

        if (disconnected)
            anInterface.showMessage("Disconnected");
        else
            anInterface.showMessage("Connected");
    }
}
```

9. Make the MainActivity Implement the interface MyBroadcaseInterface
10. Change the text of the textview with the message received

HOW TO CREATE NOTIFICATIONS

1. Create an application and name it “SimpleNotificationApp”
2. Create the Application Class
 - a. Create a class and call it **MyApp**
 - b. Extend the Application class
 - c. Make the manifest application name refer to this class. This will make the class execute as soon as your application starts. Inside this class we will create the different channels

```
<application
    android:name=".MyApp"
    android:allowBackup="true"
```

3. Create two channel IDs inside the **MyApp** class

```
public static final String MAIN_CHANNEL = "main_channel";
public static final String SECONDARY_CHANNEL = "secondary_channel";
```

4. Create the two channels

```
NotificationChannel mainChannel = new NotificationChannel(
    MAIN_CHANNEL,
    name: "MAIN CHANNEL",
    NotificationManager.IMPORTANCE_HIGH
);
```

```
mainChannel.setDescription("This is the main channel where we post .....");
```

5. Do the same for the secondary channel

```
NotificationChannel secondaryChannel = new NotificationChannel(
    SECONDARY_CHANNEL,
    name: "SECONDARY CHANNEL",
    NotificationManager.IMPORTANCE_LOW
);

//once created you can not change the behaviour , to change to install/uninstall
secondaryChannel.setDescription("This is the secondary channel where we post ...");
```

6. Create an array list of the channels

```
//Notification Manager to create the channels
ArrayList<NotificationChannel> channels = new ArrayList<>();
channels.add(mainChannel);
channels.add(secondaryChannel);
```

7. Add the channels to the notification channels using the Notification Manager service

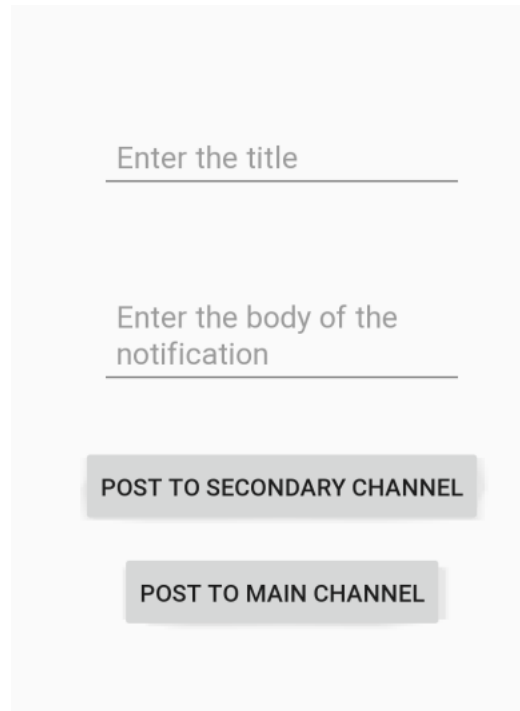
```
NotificationManager manager = getSystemService(NotificationManager.class);
manager.createNotificationChannels(channels);
```

8. Using the Notification Manager service add the channels you created to the notification channels
9. Open your main activity and try to create notification using the two channels that you have created
10. Declare the NotificationManagerCompat and instantiate it inside the onCreate method

```
//can not create channel but help us with old ones
NotificationManagerCompat notificationManagerCompat;
```

```
notificationManagerCompat = NotificationManagerCompat.from(this);
```

11. Design the following simple layout, that we can use to test out notifications



12. Add a ClickListener to both the buttons and link them to two different callback methods .
Call this methods [mainChannel, secondaryChanne;]
13. Create two icons inside your drwables to use them as the small icons for the notification.
14. In the mainChannel add the following code

```
Intent intent = new Intent( action: "com.cmps312.myownbroadcast");
PendingIntent pendingIntent = PendingIntent.getActivity( context: this,
    requestCode: 101, intent, PendingIntent.FLAG_UPDATE_CURRENT);

Notification notification = new NotificationCompat
    .Builder( context: this, MAIN_CHANNEL)
    .setContentTitle(title)
    .setSmallIcon(R.drawable.ic_looks_one_black_24dp)
    .setContentText(body)
    .setContentIntent(pendingIntent)
    .setPriority(NotificationCompat.PRIORITY_HIGH)
    .setCategory(NotificationCompat.CATEGORY_MESSAGE)
    .build();

notificationManagerCompat.notify( id: 101, notification);
```

15. Do the same for the secondary channel
16. Test your code

PRACTICE - PART A

Create the following two apps that communicate through broadcast receiver. You should follow the steps below to achieve the required tasks. Each of the step will help you understand more about the concepts we covered in the lecture and lab.

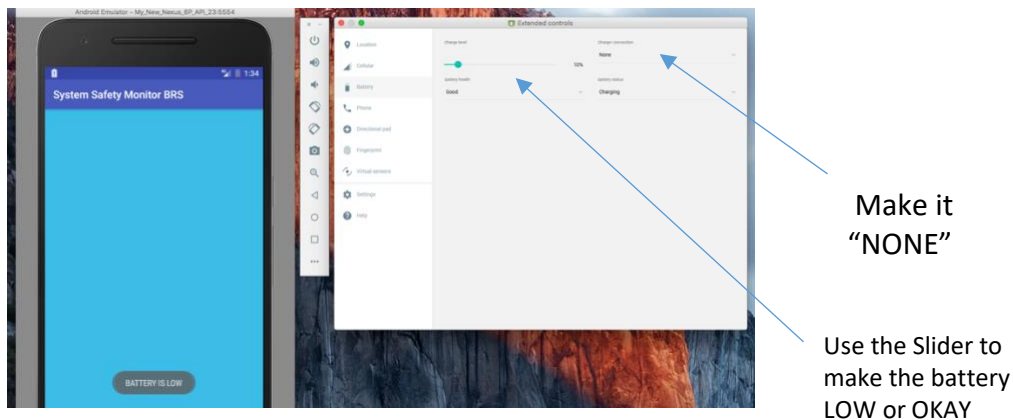
REGISTERING BROADCAST RECEIVERS STATICALLY

1. This application will listen to the **level of the system battery**.
2. In the application you have to define a broadcast receiver that displays
 - a. **“The System BATTERY is LOW”** when the system battery is **LOW**, and
 - b. **“The System BATTERY is OKAY”** when system battery is **OKAY**.
3. The display of the messages should happen inside the broadcast receiver. You can use **TOAST** or **LOG** to display the message.

NOTE: The broadcast receiver should work even when your application is **not in the foreground**.

Instructions

- A. Create an application and name it **“System Safety Monitor BRS”**
- B. Use the following domain name for the package **“CMPS312.qu.edu.qa”**
- C. Create a broadcast receiver and name it **“BatteryStatusReceiver”**
- D. Inside the broadcast receiver check if
 - a. **intent.getAction() == Intent.ACTION_BATTERY_OKAY** OR
 - i. Display **“The System BATTERY is OKAY”**
 - b. **intent.getAction() == Intent.ACTION_BATTERY_LOW**
 - i. Display **“The System BATTERY is LOW”**
- E. Register the broadcast Receiver in the Manifest
- F. Make your broadcast receiver listen to the following two actions in the manifest
 - a. **android.intent.action.BATTERY_LOW**
 - b. **android.intent.action.BATTERY_OKAY**
- G. **Run the application and Use the Emulator to test the battery status**



PART B

REGISTERING BROADCAST RECEIVERS DYNAMICALLY

1. This application will listen to a custom broadcast with the following action string **"qa.edu.cmpps312.safeToUse"**.
2. In its **BroadcastReceiver** class it expects an **EXTRA Boolean Value** passed through the Intent called **"isSafe"**.
3. If the EXTRA Boolean value in **isSafe == true** then the application Displays a Message saying **"The system is safe to use"**.
4. Otherwise it displays **"The system is NOT safe to use"**.

Instructions

- A. Create an application and name it **"Mission Control BRD"**
- B. Use the following domain name for the package **"CMPS312.qu.edu.qa"**
- C. Create a second activity and name it **"WelcomeActivity"**.
- D. In the layout of the Welcome Activity write some text saying **"Welcome to Mission Control"**. Make the Text view **Center Vertical and Horizontal**
- E. Now go to the **MainActivity** Layout and Add a button.
 - a. Make the button text **"Open Mission Control Activity"**
 - b. Make the button In-Active by making **android:enabled="false"**.
 - c. **android:onClick= "openMissionControl"** -> Create this method inside your MainActivity
- NOTE:** This button becomes active when the application is in safe mode (**It is for later use**)
- F. Create a broadcast receiver and name it **"SafetyStatusReceiver"**
- G. Inside the broadcast receiver check if **intent.getBooleanExtra("safe",false);**
 - a. If the returned value is true, then
 - i. Display **"The system is safe to use"**.
 - ii. Otherwise display **"The system is NOT safe to use"**.
- H. Create a Dynamic Broadcast **SafetyStatusReceiver** object in your MainActivity
- I. Make the intent filter action **"qa.edu.cmpps312.safeToUse"**
- J. Register the broadcast Receiver in the **Resume() Method** and,
- K. Unregister the broadcast Receiver in the **onPause() Method** in the MainActivity
- L. To test this application, we need to link it to the application that we have created in **PART A**

PART C

SENDING BROADCAST

In this part you will send a custom broadcast with the following action string **"qa.edu.cmpps312.safeToUse"**. This broadcast will be handled by the **"Mission Control BRD"** application that you have created in PART B. You also need to pass a single Boolean Value called **safe**. This value becomes true if the Battery is in OKAY status and False otherwise.

Instructions

- A. Go to your “**BatteryStatusReceiver**”
- B. Declare an implicitly intent action string “**qa.edu.cmps312.safeToUse**”.
- C. Now create a Boolean called **safe** and make it false;
- D. If battery status is **OKAY** then Make the Boolean **safe** = true
 - a. Add the **safe** value to the intent using **putExtra()** method
 - b. Broadcast the intent using **context.sendBroadcast()**
- E. Now **Run** both applications and make sure the “**Mission Control BRD**” is on the **foreground**.
- F. Then Change the battery from low to OKAY. Now both apps should show their Message.

PART D **SHOWING NOTIFICATION**

In this part you will create a Notification that shows up when the battery becomes **OKAY**. Even if the user is not currently viewing the application. Your task is to launch a notification once the battery status becomes **OKAY**. After the Notification is launched, then if the user clicks on the Notification, they should be directed to the “**Mission Control BRD**” app’s **WelcomeActivity** Screen.

PART E **COMMUNICATION BETWEEN THE ACTIVITY AND BROADCAST RECEIVER** **USING OBSERVER INTERFACE**

In this part your task is to **enable/disable** the button that you created in the “**Mission Control BRD**” app inside the **MainActivity** layout. Depending on the Battery State make the “**Open Mission Control Activity**” Button Enabled, when the application is in **SAFE** state. And make it disabled if the Application is not in safe state.

This means, when the button is enabled the user will be able to click on the button to navigate to the **WelcomeActivity** Screen. However, as soon as the battery becomes **LOW**, the Button should turn back **In-Active** restricting the user from accessing the app.
