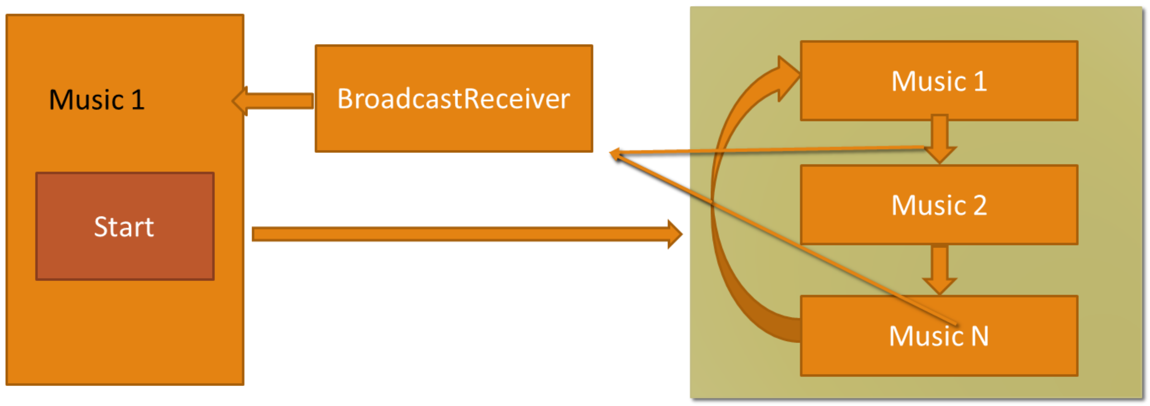
Music player:

- When the user click start, play a music
- The music keeps playing after the locking the screen or going to the home screen.
- The start button changes to pause after the music is playing, and change to resume after the music pauses.



1. Add two .mp3 files in the Music folder.
2. Create an empty project.
3. In the activity_main.xml, add a TextView on the top, and a button in the middle.

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/music"
        android:text="music"
        android:textSize="30dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center"
        android:text="Play"
        android:textSize="30dp"
        android:id="@+id/play"/>
</FrameLayout>
```

4. Create references in the MainActivity and set the OnClickListener. When click the button, the music starts, and shows the name of the music.

5. Create a MusicService class which extends Service and override the onBind method. The MusicService will run in the background to play the music.

```
public class MusicService extends Service {
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

6. Create a BroadcastReceiver classes called MusicCompletionReceiver. This service receives the event when one music starts, that is when we need to update the name of the music in the UI. The MusicCompletionReceiver extends BroadcastReceiver and override the onReceive method.

```
public class MusicCompletionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

    }
}
```

7. Now we can add the permissions, services and the BroadcastReceivers to the manifest file.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

<service android:name=".MusicService"
    android:label="Music Service"/>
<receiver android:name=".MusicCompletionReceiver"/>
```

8. Create a MusicPlayer class, which we use to control the music play. Create two static string array to save the music file paths and the music name.

```
public class MusicPlayer {
    static final String[] MUSICPATH=new String[]{
            "/Music/mario Here we go.mp3",
            "/Music/tetris.mp3",
    };
    static final String[] MUSICNAME=new String[]{
            "Super Mario",
            "Tetris",
    };
}
```

9. MediaPlayer load, initialize and play the music. Create a global MediaPlayer object. We also need a currentPosition integer to save the playback position of the music when we pause it, an integer to save the currently played music, and a musicStatus variable to save the state of the music. Create a getter method for the musicStatus. The MainActivity will use this variable to decide the status of the music. Add a getMusicName method to let the activity know the name of the music that is currently playing.

```
MediaPlayer player;
int currentPosition=0;
int musicIndex=0;
private int musicStatus=0;
public int getPlayingStatus(){
    return musicStatus;
}
public String getMusicName(){
    return MUSICNAME[musicIndex];
}
```

10. Create three methods to control the music: playMusic() will initialize and play the music, pauseMusic will pause the music and the resumeMusic to resume the current music.

```
public void playMusic() {
}

public void pauseMusic() {
}

public void resumeMusic() {
}
```

11. In the playMusic method, we initialize the MediaPlayer, and play the music by calling start() method.

```
public void playMusic(){
    try {
        player=new MediaPlayer();
        player.setDataSource(Environment.getExternalStorageDirectory() +
MUSICPATH[musicIndex]);
        player.setAudioStreamType(AudioManager.STREAM_MUSIC);
        player.prepare();
        player.start();
        musicStatus=1;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

In the pauseMusic method, we pause the music, and save the current playback position. In the resumeMusic method, we go back to that position and resume the music.

```
public void pauseMusic() {
    if (player != null && player.isPlaying()) {
        player.pause();
        currentPosition = player.getCurrentPosition();
        musicStatus = 2;
    }
}

public void resumeMusic() {
    if (player != null) {
        player.seekTo(currentPosition);
        player.start();
        musicStatus = 1;
    }
}
```

12. The music player will play the next music when finished. Add a OnCompletionListener to the MusicPlayer class, and override the onCompletion method. In the onCompletion method, we play the next music in the array.

```
MusicPlayer implements MediaPlayer.OnCompletionListener
```

```
@Override
public void onCompletion(MediaPlayer mp) {
    musicIndex=(musicIndex+1)%MUSICNAME.length;
    player.release();
    player=null;
    playMusic();
}
```

In the playMusic method, register the listener.

```
player.setOnCompletionListener(this);
```

13. In the MusicService class, create a MusicPlayer object and initialize it in the onCreate method. Add the start, pause, resume and getPlayingStatus methods to control the music.

```
MusicPlayer musicPlayer;

@Override
public void onCreate() {
    super.onCreate();
    musicPlayer = new MusicPlayer();
}

public void startMusic() {
    musicPlayer.playMusic();
}

public void pauseMusic() {
    musicPlayer.pauseMusic();
}

public void resumeMusic() {
    musicPlayer.resumeMusic();
}

public int getPlayingStatus(){
    return musicPlayer.getPlayingStatus();
}
```

14. Create 2 static Strings in MusicService. One serves as the BroadcastReceiver identifier and another as the music name identifier.

```
public static final String COMPLETE_INTENT = "complete intent";
public static final String MUSICNAME = "music name";
```

15. Create an onUpdateMusicName method to broadcast the complete event. Use the Intent to send broadcast to update the UI. It is used to notify the attached activity to update the music name, triggered when the music is ready to play. We call the method in the startMusic().

```
public void onUpdateMusicName(String musicname) {
    Intent intent = new Intent(COMPLETE_INTENT);
    intent.putExtra(MUSICNAME, musicname);
    sendBroadcast(intent);
}
```

16. The MusicPlayer will work in the MusicService class, so we add a global variable for the service, and initialize it in the constructor.

```
private MusicService musicService;

public void setMusicService(MusicService musicService) {
    this.musicService = musicService;
}
```

In the onCreateMethod of MusicService, initialize the musicservice.

```
@Override
public void onCreate() {
    super.onCreate();
    musicPlayer=new MusicPlayer();
    musicPlayer.setMusicService(this);
}
```

Every time a new music plays, we broadcast the completion event. In the MusicPlayer class, call the onUpdateMusicName method

```
public void playMusic() {
    try {
        musicService.onUpdateMusicName(getMusicName());
```

17. In the broadcast receiver, we handle the broadcast event. The broadcast receiver will update the MainActivity, so it needs a ref to it. Create a MainActivity reference and initialize it in the constructor.

```
MainActivity mainActivity;
public MusicCompletionReceiver(MainActivity mainActivity){
    this.mainActivity=mainActivity;
}
```

In the MusicCompetionReceiver, we retrieve the music name, and update the TextView.

```
@Override
public void onReceive(Context context, Intent intent) {
    String musicName=intent.getStringExtra(MusicService.MUSICNAME);
    mainActivity.updateName(musicName);
}
```

Add a method in the MainActivity to update the TextView.

18. In the MainActivity, create a MusicService, an intent to start and bind the service and a MusicCompetionReceiver. Create two Boolean values to record whether the service has been initialized and been binded.

```
MusicService musicService;
MusicCompletionReceiver musicBroadcastReceiver;
Intent startMusicServiceIntent;
boolean isInitialized=false;
boolean isBound=false;
```

Initialize them in the onCreate Method.

```
startMusicServiceIntent = new Intent(this, MusicService.class);

if (!isInitialized) {
    startService(startMusicServiceIntent);
    isInitialized = true;
}

musicBroadcastReceiver = new MusicCompletionReceiver(this);
```

19. In the onClick method, based on which state the music is at, we call different music controlling method in the music service

```
@Override
public void onClick(View v) {
    if(isBound){
        switch (musicService.getPlayingStatus()){
            case 0:
                musicService.startMusic();
                play.setText("Pause");
                break;
            case 1:
                musicService.pauseMusic();
                play.setText("Resume");
                break;
            case 2:
                musicService.resumeMusic();
                play.setText("Pause");
                break;
        }
    }
}
```

20. We need a Binder to let the activity attach to the MusicService and thereafter control the music. Create a private MyBinder class in MusicService class. In the onBind method, return the iBinder.

```
private final IBinder iBinder = new MyBinder();
@Nullable
@Override
public IBinder onBind(Intent intent) {
    return iBinder;
}

public class MyBinder extends Binder {
    MusicService getService() {
        return MusicService.this;
    }
}
```

21. Create a private ServiceConnection varible in the MainActivity. This class is used to bind the activity to the service. Once bound, we retrive the service.

```
private ServiceConnection musicServiceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        MusicService.MyBinder binder = (MusicService.MyBinder) service;
        musicService = binder.getService();
        isBound = true;
        switch (musicService.getPlayingStatus()) {
            case 0:
                play.setText("Start");
                break;
            case 1:
                play.setText("Pause");
                break;
            case 2:
                play.setText("Resume");
                break;
        }
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        musicService = null;
        isBound = false;
    }
};
```

22. Now we have to deal with the activity lifecycle events. For the service, we just want start it once. Remember we use a isInitialized indicator to record whether the service has been initialized. But when the activity is paused, this value will be set to default. So we have to save it in the in the onSaveIntanceState method. We also want show the music name in after the screen rotates, so save the music name here too.

```
public static final String INITIALIZE_STATUS = "intialization status";
public static final String MUSIC_PLAYING = "music playing";

@Override
protected void onSaveInstanceState(Bundle outState) {
    outState.putBoolean(INITIALIZE_STATUS,isInitialized);
    outState.putString(MUSIC_PLAYING,music.getText().toString());
    super.onSaveInstanceState(outState);
}
```

In the onCreate method, we retrieve those data from the savedInstanceState.

```
if(savedInstanceState!=null){
    isInitialized=savedInstanceState.getBoolean(INITIALIZE_STATUS);
    music.setText(savedInstanceState.getString(MUSIC_PLAYING));
}
```

23. In the onResume() method, we bind the service, and register the BroadcastReceiver.

```
@Override
protected void onResume() {
    super.onResume();
    if (isInitialized && !isBound) {
        bindService(startMusicServiceIntent, musicServiceConnection,
Context.BIND_AUTO_CREATE);
    }
    registerReceiver(musicBroadcastReceiver,new
IntentFilter(MusicService.COMPLETE_INTENT));
}
```

In the onPause() method, we unbind the service, and unregister the BroadcastReceiver.

```
@Override
protected void onPause() {
    super.onPause();
    if (isBound) {
        unbindService(musicServiceConnection);
        isBound = false;
    }
    unregisterReceiver(musicBroadcastReceiver);
}
```