**BAB VI**

***Service***

* **Tujuan**
* Mahasiswa dapat mengetahui definisi dan manfaat service.
* ***Software* yang dibutuhkan**
* *Java JDK*
* *Android Studio 3.5*
* *SDK API 29*

**6.3 Teori**

Kita telah belajar bagaimana mengenai *Activity* dan implementasinya. *Activity* dan *Fragment* adalah dua komponen yang memberikan pengalaman kepada praktikan secara langsung. Praktikan dapat melihat dan berinteraksi diatasnya.

*Service* berada pada sisi yang lain. Komponen ini tidak memiliki antarmuka dan bahkan pengguna tidak akan tahu bagaimana dia bekerja. Pengalaman yang diberikan oleh *service* hanya berupa proses yang tidak terlihat secara *background* untuk melakukan berbagai macam proses yang memakan waktu lama. Walaupun berjalan secara *background,* pada dasarnya *Service* dan komponen *Android* lainnya berjalan pada suatu proses dan *thread* yang sama yaitu *mainthread* atau *UIThread*. Bekerja secara *background* bukan berarti *service* bisa bekerja secara *Asynchronous*, akan tetapi tetap membutuhkan *thread* terpisah jika kita ingin melakukan proses yang bersifat komputasi intensif atau yang memakan waktu.

Contoh pemanfaatan *Service* sudah banyak sekali, beberapa diantarnya :

* Aplikasi *social media* atau aplikasi yang memiliki kemampuan untuk menerima *pushnotification*, pasti memiliki sebuah *service* yang berjalan dalam posisi *stand by* untuk selalu menerima pesan yang masuk atau juga untuk men*download file*.
* Aplikasi *chatting* juga membutuhkan *service* untuk melakukan pengiriman dan menerima pesan yang dikirimkan oleh pengguna.
* Aplikasi pendengar musik juga melakukan hal yang sama. Untuk memberikan pengalaman yang lebih baik kepada pengguna, aplikasi pendengar musik biasanya meletakkan proses *streaming* atau memainkan musik di komponen *Service* dengan tetap mempertahankan integrasi dengan komponen lain misalnya seperti notifikasi.

Secara umum terdapat dua bentuk dari *Service*, yaitu :

* *Started Service*

*Service* berjenis ini adalah tipe yang dijalankan oleh komponen lain, misal *Activity*. Sekali dijalankan, *service* ini akan berjalan selama belum dimatikan atau proses yang dijalankan selesai. Walaupun komponen yang lain di *stop* atau dimatikan oleh *Android*, sistem atau *Activity* sudah di *destroy*, *Service* akan tetap berjalan. Umunya penggunaan *service* ini adalah untuk melakukan proses yang tidak memberikan nilai balik ke komponen yang memanggil. Contohnya, adalah men*download* atau meng*upload file* dari dan ke *server*.

* *Bound Service*

*Service* jenis ini merupakan tipe *service* yang dijalankan oleh komponen lain, namun saling mengikat satu sama lain. Hubungan yang terjadi antar kedua komponen tersebut seperti *client-server*. Bisa saling menerima hasil dan menerima *request* yang ada. Pada *service* ini dimungkinkan terjadi proses *IPC (Interprocess Communication)*. *Service* ini akan tetap hidup secara *background* selama masih ada komponen lain yang mengikatnya. Jika tidak, maka *Service* akan dimatikan oleh sistem. Aplikasi pemutar musik merupakan salah satu jenis aplikasi yang mengimplementasikan *Service* jenis ini.

Pada bagian ini kita sepenuhnya membahas *Service* berjenis *started.* Di tipe *Service* tersebut akan dibagi menjadi dua bagian dalam implementasinya, yaitu :

* Kelas *Service* yang *inherit* langsung kepada kelas *Service.*

Ketika sebuah kelas *java inherit* ke *Service* ingin menjalankan proses yang memakan waktu lama misal men*download file*, maka diharuskan membuat *thread* terpisah agar tidak memblok *UIThread* yang ada. *Service* ini akan selalu hidup di *background* selama tidak ada komponen yang memanggil *stopService()* atau sistem yang mematikan jika dibutuhkan memori tambahan untuk menjalankan aplikasi lain.

* Kelas *Service* yang *inherit* ke *Intent Service*.

Ini adalah kelas yang sangat memudahkan hidup kita. Dia bersifat *fire and forget*, ketika ia telah menyelesaikan tugasnya, ia akan mematikan dirinya.

Praktikan akan belajar mengimplementasikan kedua tipe *service* tersebut pada bab ini.

Poin-poin penting lain tentang *Service* diantaranya adalah :

* Setiap kelas java akan dinyatakan sebagai sebuah *service* apabila kelas tersebut menerapkan *inherit/extends* ke kelas *Service* atau *IntentService*.
* *Service* memiliki *lifecycle*-nya sendiri dan bergantung kepada tipe *service* apa yang digunakan, *Started* atau *BoundService*.

Untuk menjalankan *Service* dari komponen lain seperti *Activity*, cukup menggunakan *startService(Intent)* dengan objek intent yang dikirimkan. Sebaiknya untuk mematikan/*stop service* terdapat dua cara yaitu : *stopService(Intent)* dijalankan dari komponen yang memanggil dan *stopSelf()* dari kelas *Service* itu sendiri.

**BAB VII**

*BroadcastReceiver*

**7.1 Tujuan**

* Mahasiswa dapat membuat receiver sederhana.
* Mahasiswa dapat membuat broadcast event pada broadcastreceiver.
* Mahasiswa dapat menggunakan emulator untuk

**7.2 *Software* yang dibutuhkan**

* *Java JDK*
* *Android Studio 3.5*
* *SDK API 29*

**7.3 Teori**

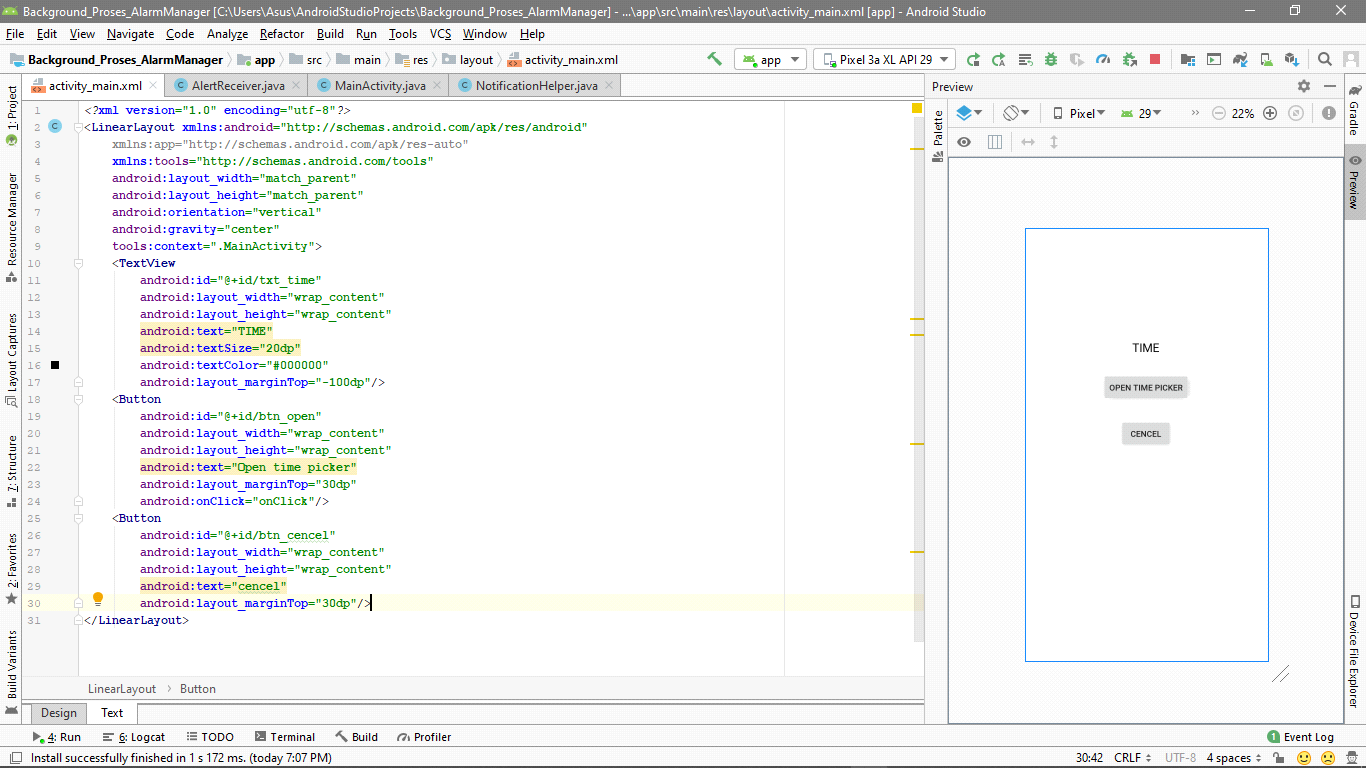
Ketika mengembangkan sebuah aplikasi di *Android*, adakalanya kita butuh menjalankan suatu proses tertentu ketika sebuah proses lain telah terjadi. Misal, kita ingin menjalankan sesuatu ketika *device Android* sudah selesai *booting* dengan sempurna. Kemampua ini dapat kita manfaatkan untuk memberi pengalaman yang lebih baik. Disinilah komponen *broadcast receiver* berperan penting. Menurut definisinya, *broadcast receiver* adalah cara sederhana untuk menindaklanjuti sebuah *broadcast message* yang dipancarkan oleh dirinya sendiri, aplikasi lain atau sistem *Android*. Sementara itu, *broadcast message* adalah pesan yang akan dipancarkan melalui *objek intent*. Saat ini banyak aplikasi yang menggunakan validasi pengguna dengan mengirimkan kode verifikasi melalui *SMS*. Tanpa disadari, ketika *SMS* tersebut masuk dan ia berasal dari *provider* aplikasi, maka akan ada komponen dari aplikasi lain yang akan menindaklanjuti dan memproses pesan tadi. Misalnya, dengan melakukan konfirmasi proses validasi tersebut dengan cara menghubungi sebuah *API* di *server*. Interaksi antara *BroadcastReceiver* dengan *Android System*.

Pada materi kali ini praktikan akan belajar bagaimana mengimplementasikan dua mekanisme umum dalam memanfaatkan komponen *BroadcastReceiver*, yaitu :

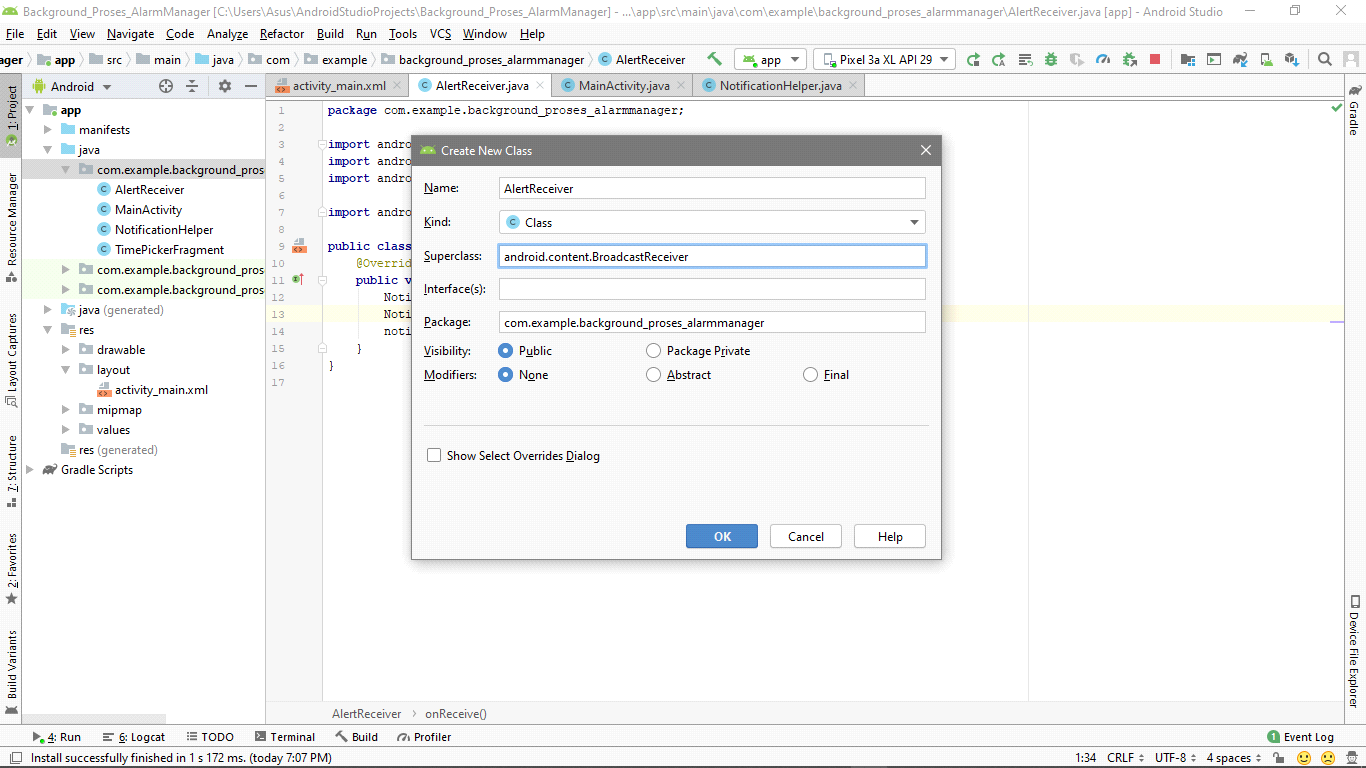
* Implementasi *BroadcastReceiver* untuk menerima *event* yang di*broadcast* oleh sistem *Android*. Praktikan akan praktik bagaimana menerima *broadcast* ketika ada *SMS* yang masuk.
* Implementasi *BroadcastReceiver* untuk komunikasi antar aplikasi.

**7.4 Praktikum**

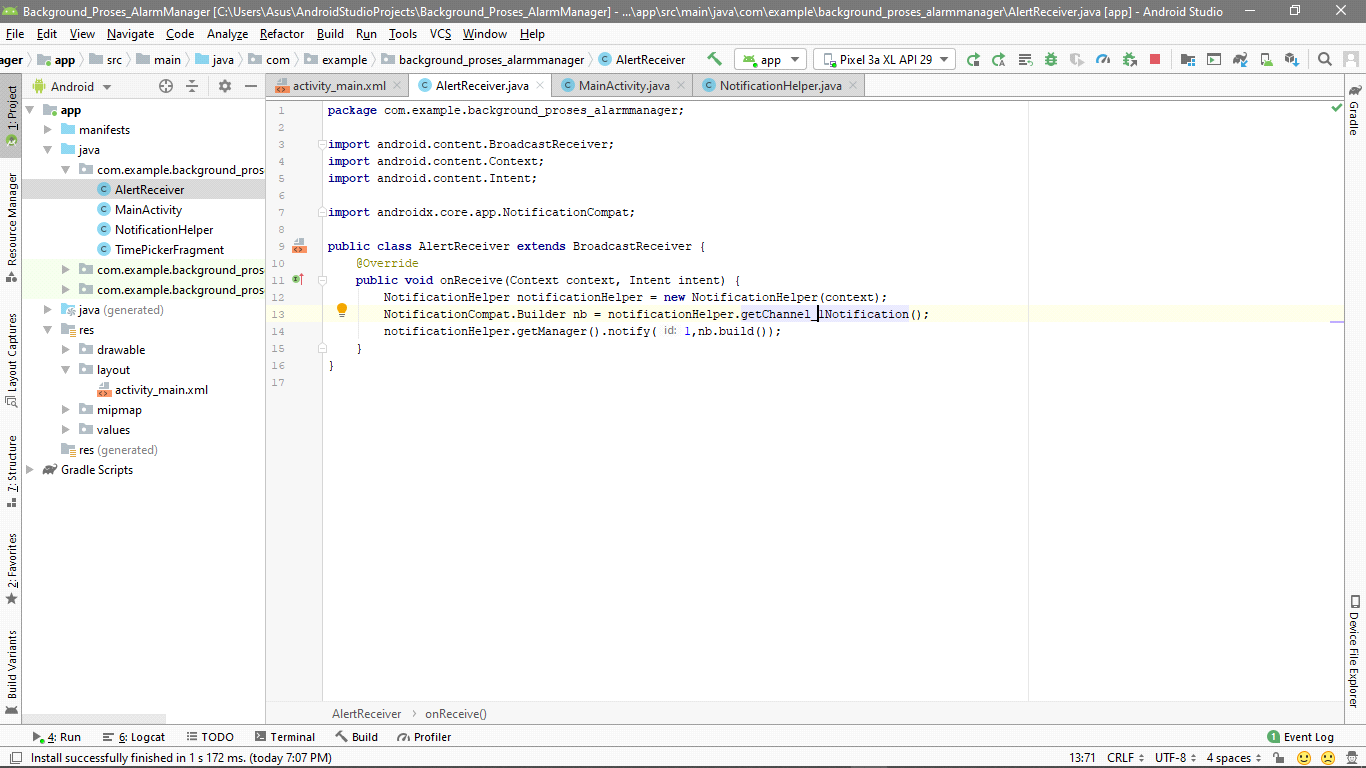
* Desa in Activity\_main.xml seperti gambar dibawah ini.



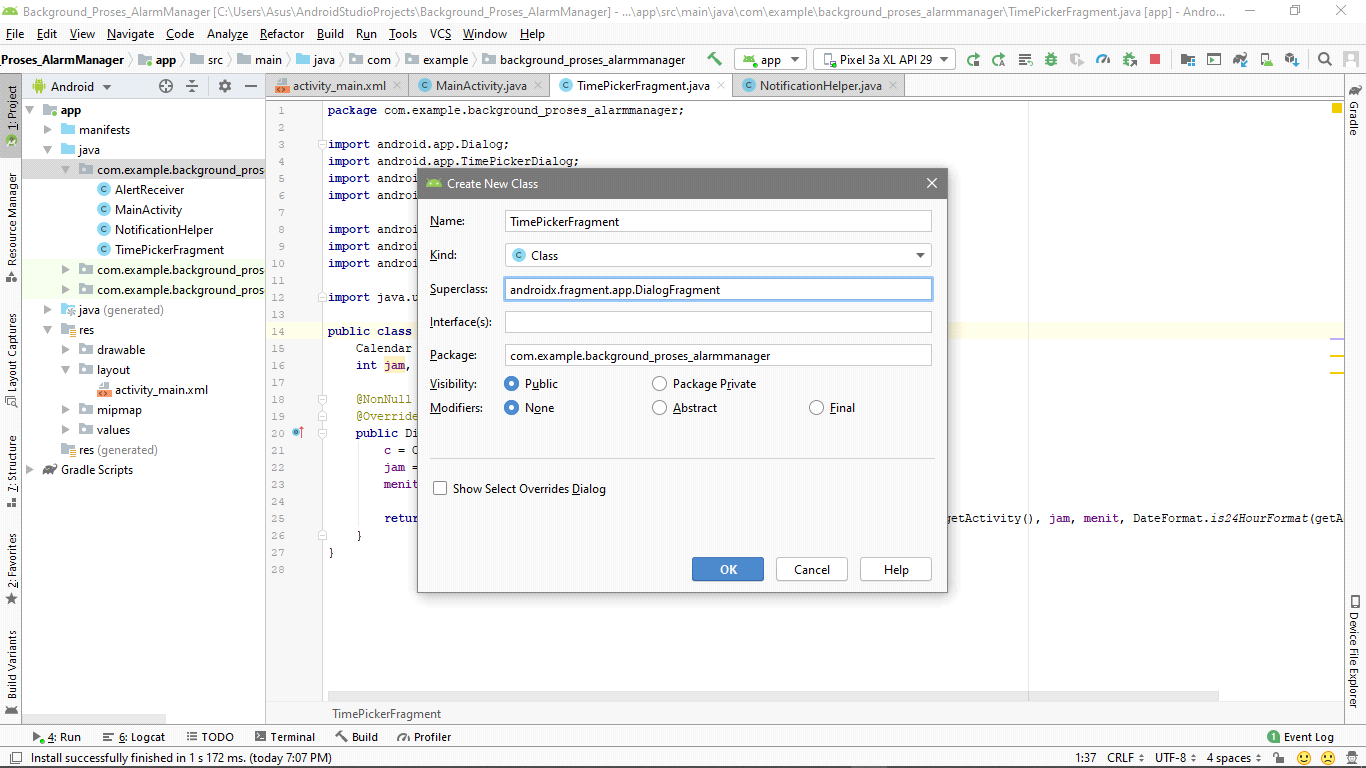
* Buat class baru dengan cara klik kanan pada packed java > new > java class, beri nama classnya “AlertReceiver” dan pada Superclassnya ketikkan BroadcastReceiver. Klik OK.



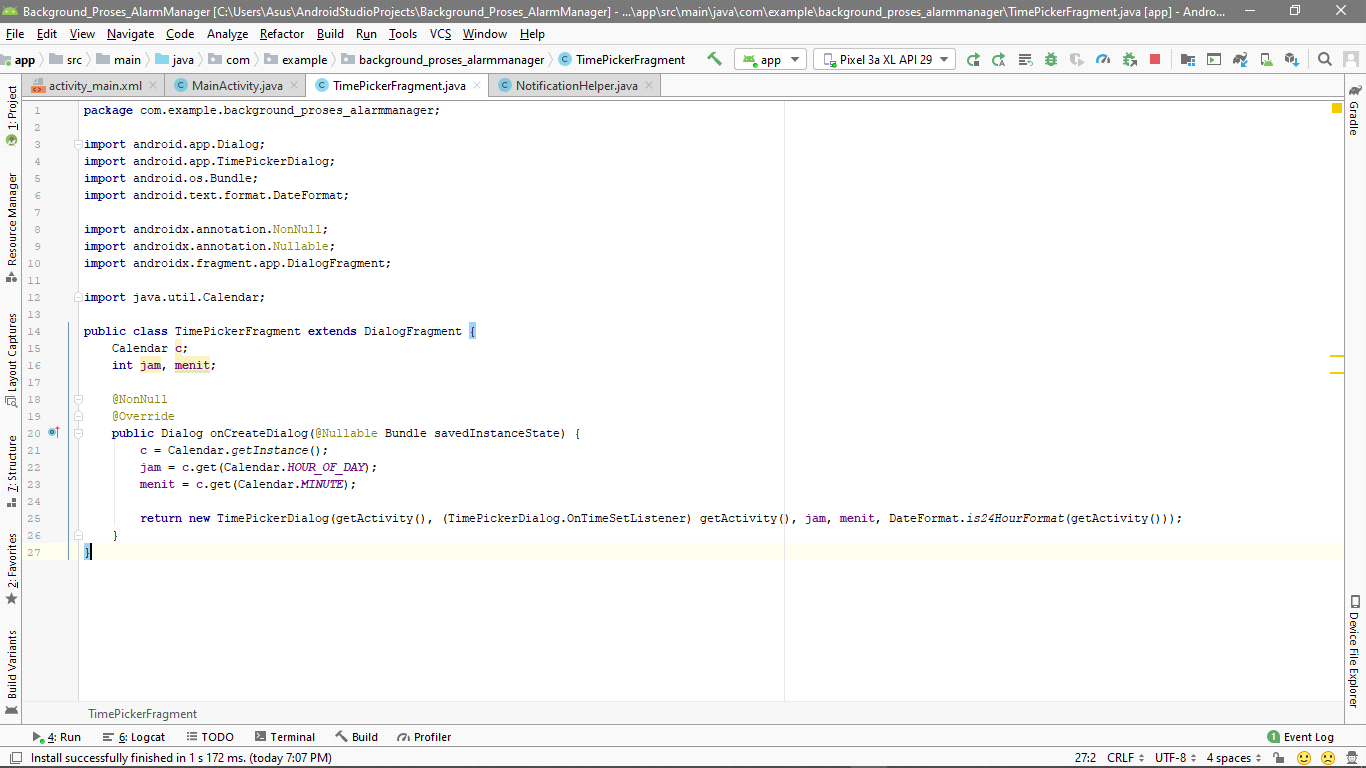
* Masukkan sourcode seperti dibawah ini.



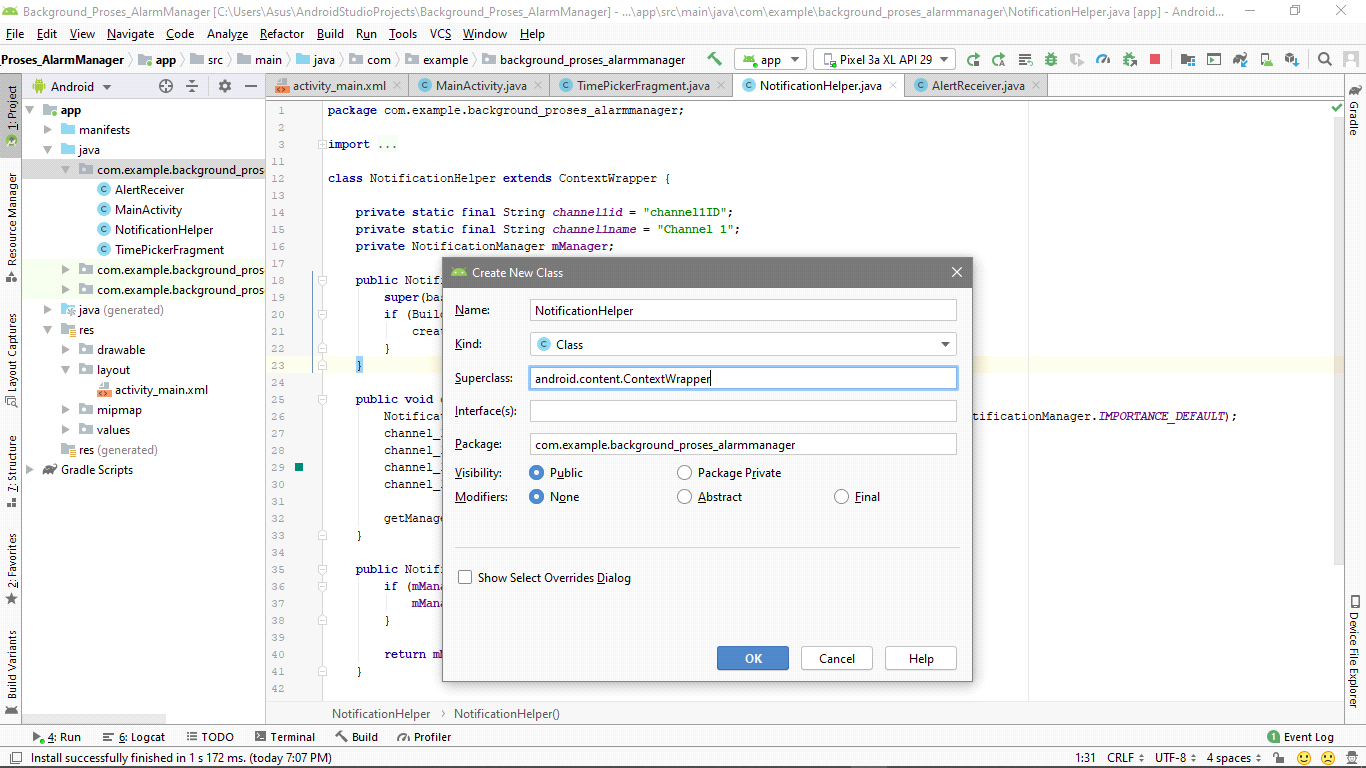
* Buat lagi java class baru, beri nama class java dengan “TimePickerFragment” dan pada Superclass ketik DialogFragment. Klik OK.



* Pada kelas TimePickerFragment masukkan sourcode seperti dibawah ini.



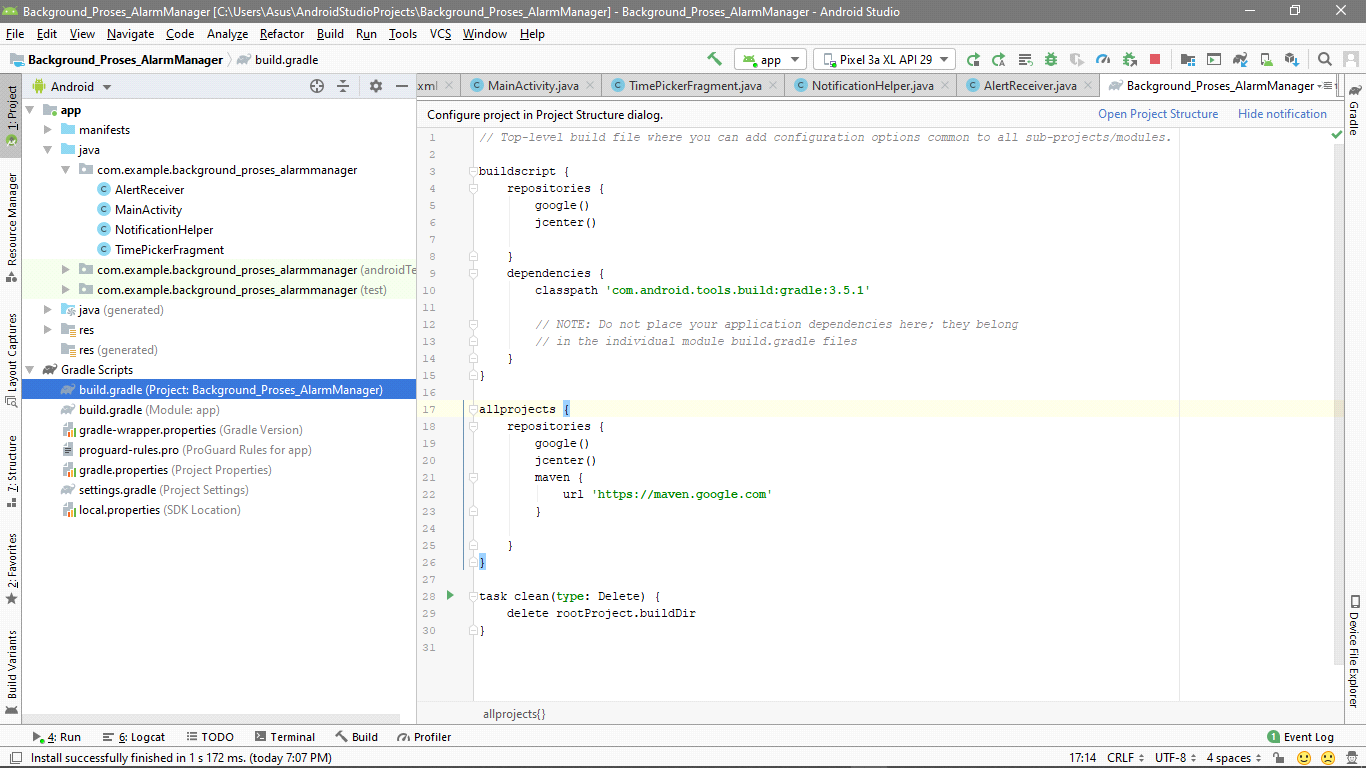
* Buat java class baru dengan nama “NotificationHelper” dan ketik pada Superclass “ContexWrapper”. Klik OK.



* Pada class NotificationHelper masukkan source code dibawah ini.

|  |
| --- |
| **package** com.example.background\_proses\_alarmmanager;  **import** android.app.Notification; **import** android.app.NotificationChannel; **import** android.app.NotificationManager; **import** android.content.Context; **import** android.content.ContextWrapper; **import** android.os.Build;  **import** androidx.core.app.NotificationCompat;  **class** NotificationHelper **extends** ContextWrapper {   **private static final** String ***channel1id*** = **"channel1ID"**;  **private static final** String ***channel1name*** = **"Channel 1"**;  **private** NotificationManager **mManager**;   **public** NotificationHelper(Context base) {  **super**(base);  **if** (Build.VERSION.***SDK\_INT*** >= Build.VERSION\_CODES.***O***){  createChannels();  }  }    **public void** createChannels(){  NotificationChannel channel\_1 = **new** NotificationChannel(***channel1id***, ***channel1name***, NotificationManager.***IMPORTANCE\_DEFAULT***);  channel\_1.enableLights(**true**);  channel\_1.enableVibration(**true**);  channel\_1.setLightColor(R.color.***colorPrimary***);  channel\_1.setLockscreenVisibility(Notification.***VISIBILITY\_PRIVATE***);   getManager().createNotificationChannel(channel\_1);  }   **public** NotificationManager getManager(){  **if** (**mManager** == **null**){  **mManager** = (NotificationManager) getSystemService(Context.***NOTIFICATION\_SERVICE***);  }   **return mManager**;  }   **public** NotificationCompat.Builder getChannel\_1Notification(){  **return new** NotificationCompat.Builder(getApplicationContext(), ***channel1id***)  .setSmallIcon(R.drawable.***ic\_satu***);  } } |

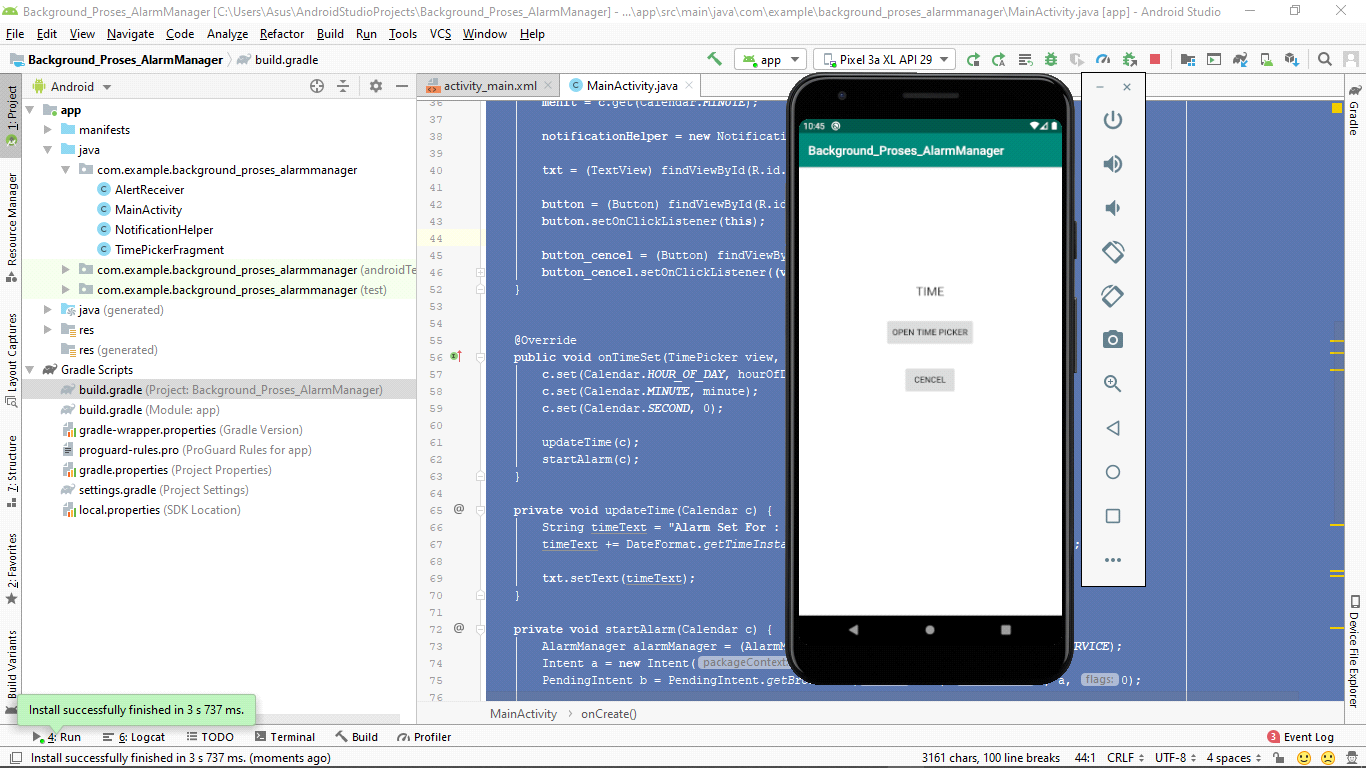
* Pada packed build.gradle(Project) tambah source code seperti pada gambar dibawah ini.



* Pada class MainActivity.java masukkan source Code Seperti dibawah ini.

|  |
| --- |
| **package** com.example.background\_proses\_alarmmanager;  **import** android.app.AlarmManager; **import** android.app.PendingIntent; **import** android.app.TimePickerDialog; **import** android.content.Context; **import** android.content.Intent; **import** android.os.Bundle; **import** android.view.View; **import** android.widget.Button; **import** android.widget.TextView; **import** android.widget.TimePicker;  **import** androidx.appcompat.app.AppCompatActivity; **import** androidx.core.app.NotificationCompat; **import** androidx.fragment.app.DialogFragment;  **import** java.text.DateFormat; **import** java.util.Calendar;  **public class** MainActivity **extends** AppCompatActivity **implements** TimePickerDialog.OnTimeSetListener, View.OnClickListener {  **private** TextView **txt**;  Button **button**, **button\_cencel**;  **private** NotificationHelper **notificationHelper**;   Calendar **c**;  **int jam**, **menit**;   @Override  **protected void** onCreate(**final** Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main***);   **c** = Calendar.*getInstance*();  **jam** = **c**.get(Calendar.***HOUR\_OF\_DAY***);  **menit** = **c**.get(Calendar.***MINUTE***);   **notificationHelper** = **new** NotificationHelper(**this**);   **txt** = (TextView) findViewById(R.id.***txt\_time***);   **button** = (Button) findViewById(R.id.***btn\_open***);  **button**.setOnClickListener(**this**);   **button\_cencel** = (Button) findViewById(R.id.***btn\_cencel***);  **button\_cencel**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  cancelAlarm();  }  });  }    @Override  **public void** onTimeSet(TimePicker view, **int** hourOfDay, **int** minute) {  **c**.set(Calendar.***HOUR\_OF\_DAY***, hourOfDay);  **c**.set(Calendar.***MINUTE***, minute);  **c**.set(Calendar.***SECOND***, 0);   updateTime(**c**);  startAlarm(**c**);  }   **private void** updateTime(Calendar c) {  String timeText = **"Alarm Set For : "**;  timeText += DateFormat.*getTimeInstance*(DateFormat.***SHORT***).format(c.getTime());   **txt**.setText(timeText);  }   **private void** startAlarm(Calendar c) {  AlarmManager alarmManager = (AlarmManager) getSystemService(Context.***ALARM\_SERVICE***);  Intent a = **new** Intent(**this**, AlertReceiver.**class**);  PendingIntent b = PendingIntent.*getBroadcast*(**this**, 1, a, 0);   alarmManager.setExact(AlarmManager.***RTC\_WAKEUP***, c.getTimeInMillis(), b);  }   **private void** cancelAlarm() {  AlarmManager alarmManager = (AlarmManager) getSystemService(Context.***ALARM\_SERVICE***);  Intent a = **new** Intent(**this**, AlertReceiver.**class**);  PendingIntent b = PendingIntent.*getBroadcast*(**this**, 1, a, 0);   alarmManager.cancel(b);  **txt**.setText(**"Alarm Canceled"**);  }   @Override  **public void** onClick(View v) {  DialogFragment timePicker = **new** TimePickerFragment();  timePicker.show(getSupportFragmentManager(), **"Time Picker"**);  }   **public void** kirimnotif(String title, String message){  NotificationCompat.Builder nb = **notificationHelper**.getChannel\_1Notification();  **notificationHelper**.getManager().notify(1, nb.build());   }  } |

* Tampilan Program.



**7.5 Tugas**

* …………