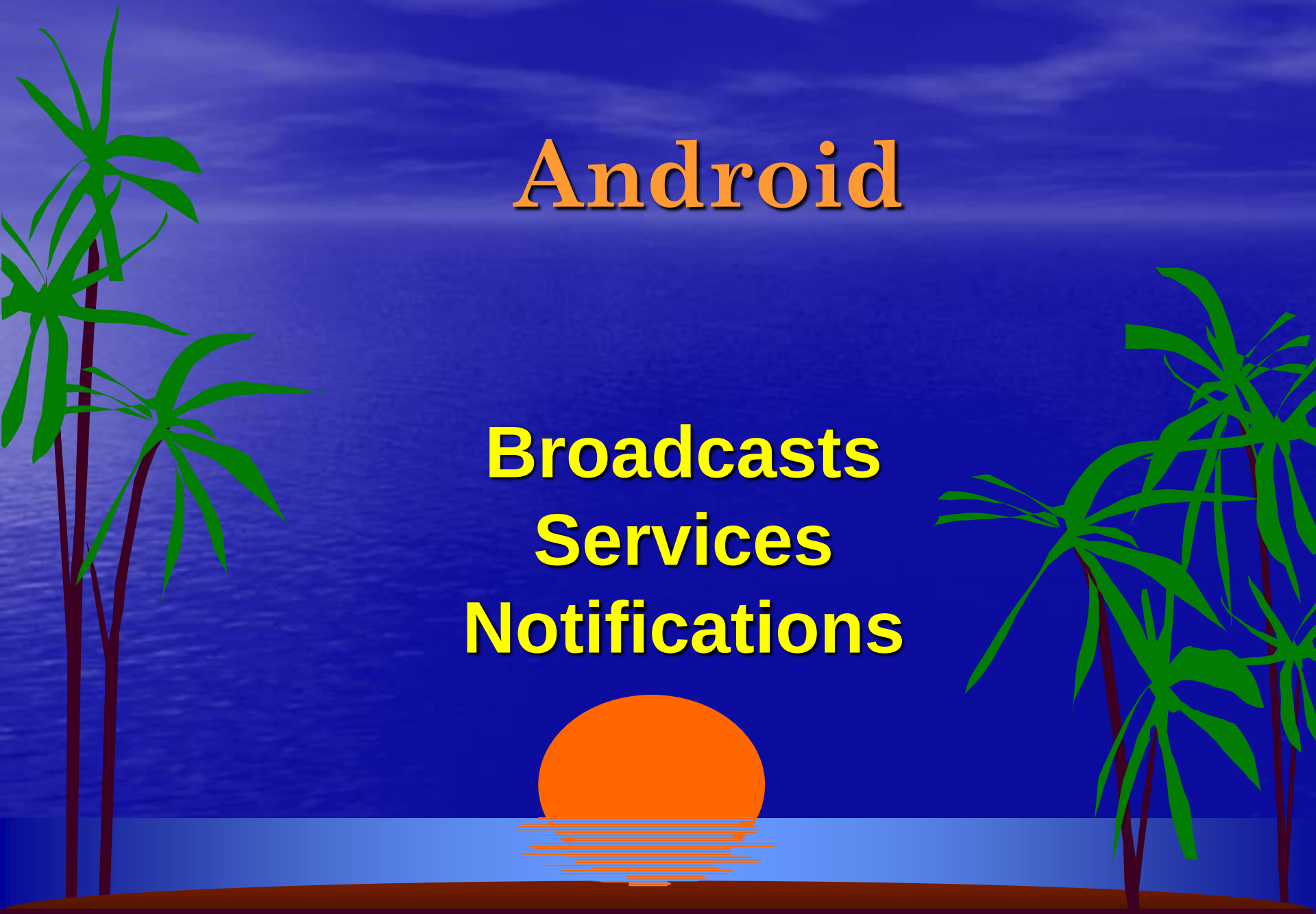
Android

Broadcasts
Services
Notifications

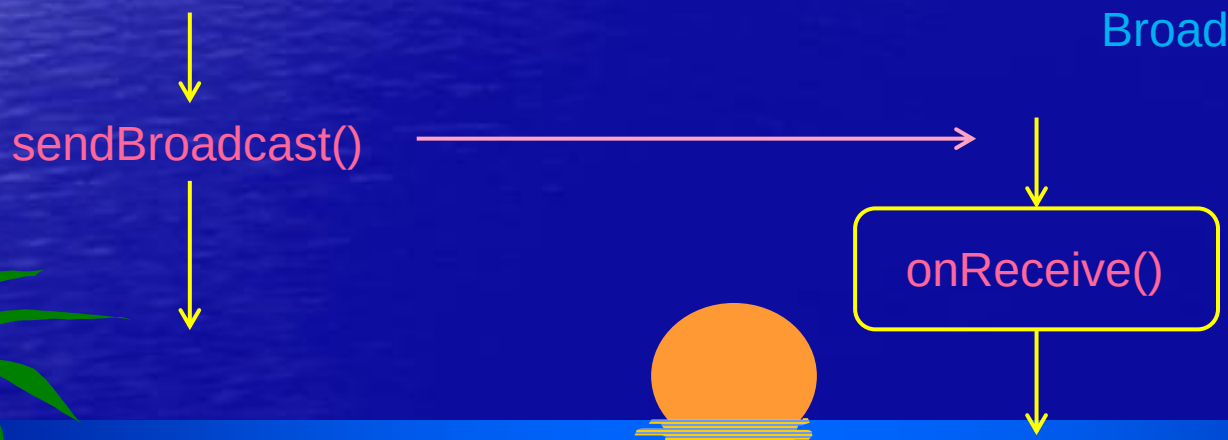


Broadcast receivers

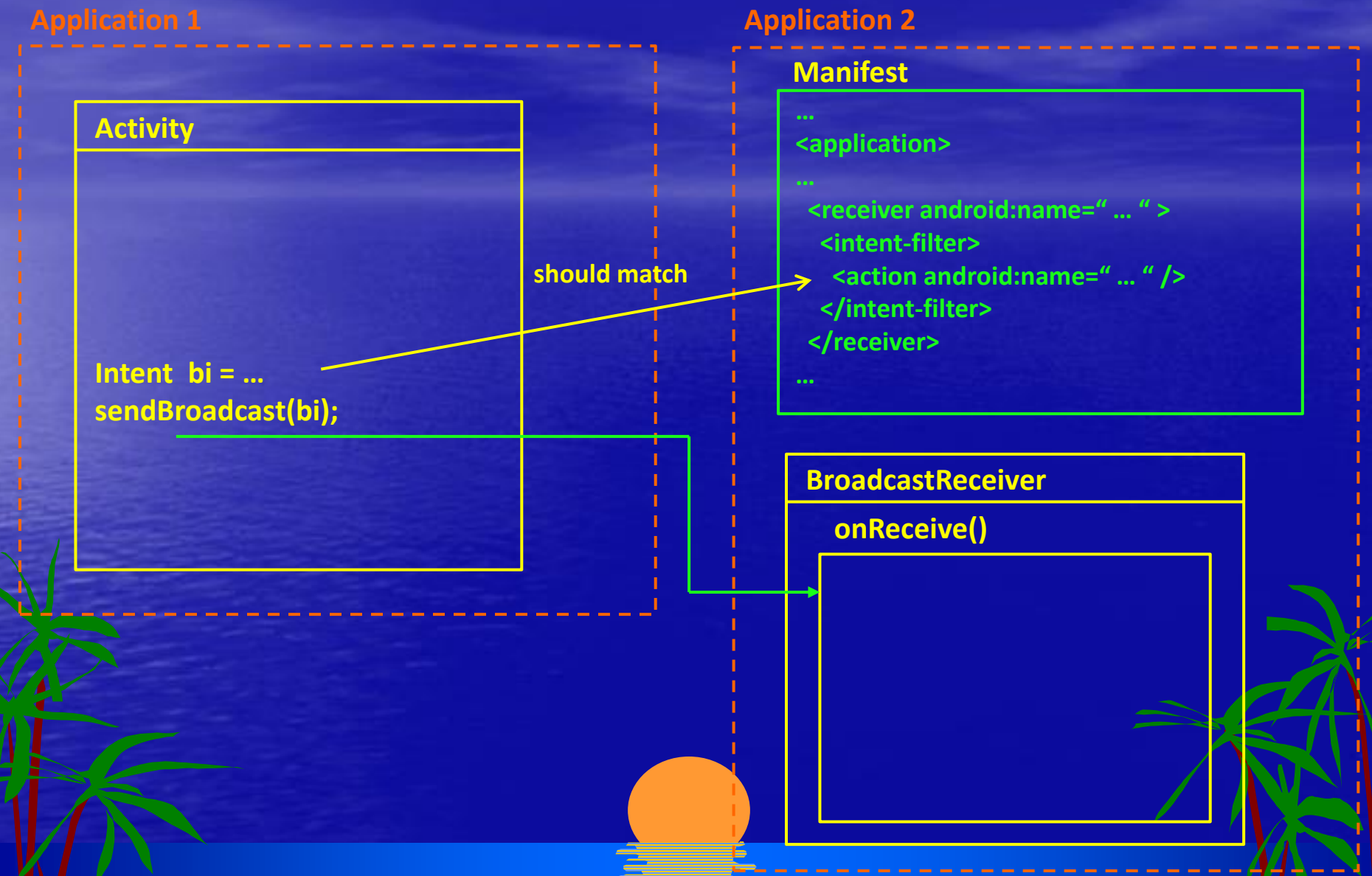
- Application components that can receive ‘intents’ from other components
  - Broadcast receivers can be declared in the manifest or registered dynamically
  - They can have an associated ACTION or cross-application explicit intent
  - They are invoked using sendBroadcast()
    - It needs an intent matching the declared one (action) or package and class name
    - The intent can transport extra data
    - sendBroadcast() can be invoked by any other application component (Activity, Service, Content Provider) in the same or other application (with restrictions after API 26)
  - Broadcast receivers extend class BroadcastReceiver
    - They must override the method onReceive()
    - They don’t have any user interface
  - The application containing the Broadcast receiver is activated and the onReceive() method invoked

Broadcast Receivers

- Receives notifications (intents) sent by other applications (mainly the by the OS components)
  - Inherits from *android.content.BroadcastReceiver*
  - Can be declared in the <receiver> tag in the Manifest
  - Can be declared programmatically (Context.registerBroadcast())
  - Normally execute in response to calls to *Context.sendBroadcast(Intent)*
  - The *onReceive(context, intent)* method executes



Sending a broadcast



Broadcast receiver example

The receiver

```
public class MyReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
         String msg = intent.getStringExtra("somename");
         //Do something
   }
}
```

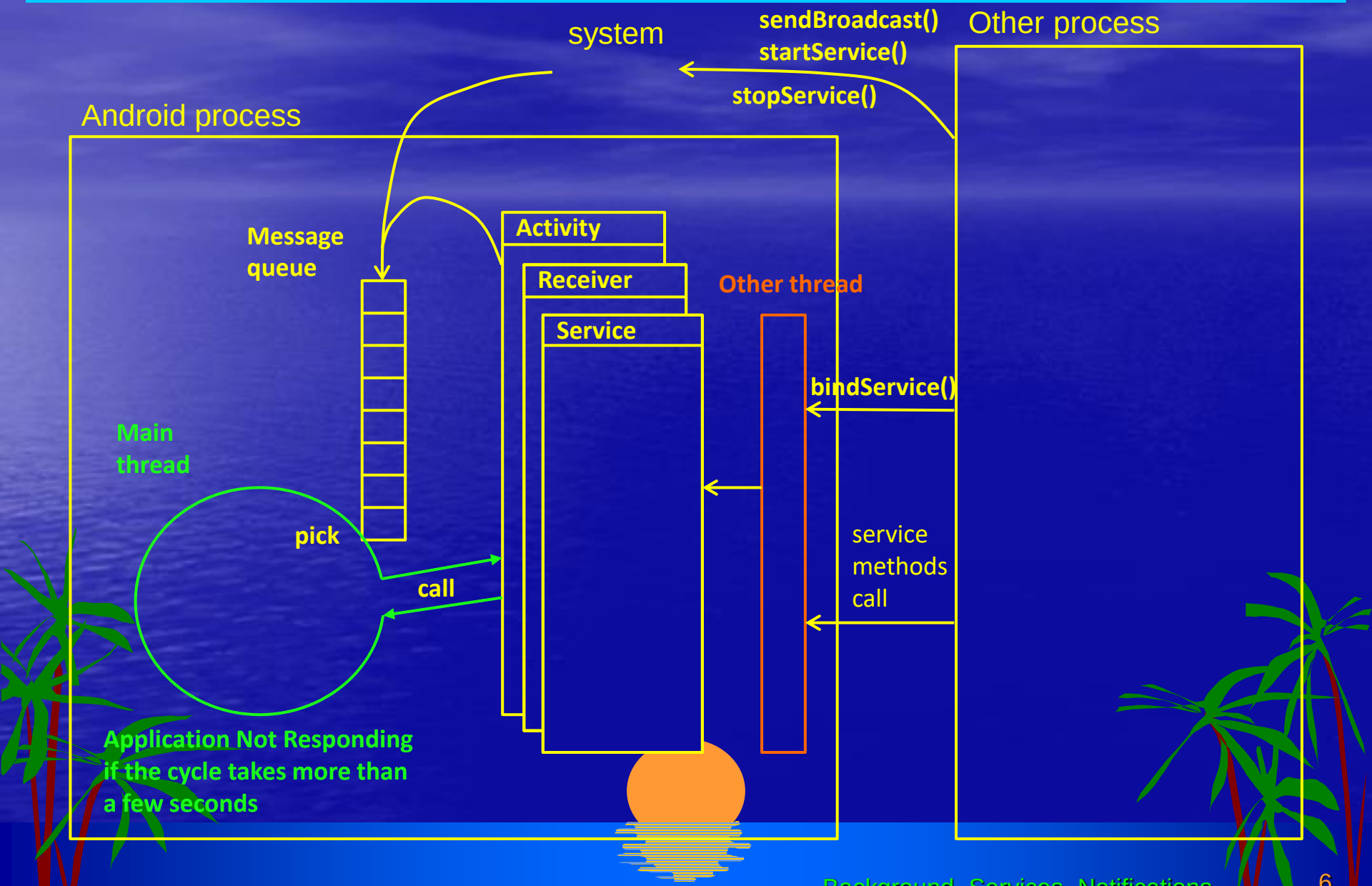
Manifest definition

```
<manifest>
 <application>
   ...
   <receiver android:name=".MyReceiver">
    <intent-filter>
      <action android:name="org.feup.intents.test" />
    </intent-filter>
   </receiver>
   ...
 </application>
 ...
</manifest>
```

The broadcast Activity

```
public class MyActivity extends Activity {
   ...
   private void invokeReceiver() {
      Intent broadcast = new Intent (
                                    "org.feup.intents.test");
      broadcast.putExtra("somename", "Hello");
      sendBroadcast(broadcast);
   }
   ...
}
```

Processes and receivers / services



Services

- Can be invoked from other clients
  - Clients are in the same process or in other processes
    - Using a local intent (class) or an implicit one (action)
  - Services don´t have a user interface
  - If the service process is not in memory, it is started
    - the onCreate() method is executed
  - Any client can invoke a service asynchronously
    - calling startService() which will invoke onStartCommand()
    - stopService() will try to terminate the service (onDestroy() is invoked in this procedure)
    - A service can terminate itself calling stopSelf()
  - A client can call bindService() to establish a channel and obtain a service interface (remote call service)
    - The client can then call the interface methods

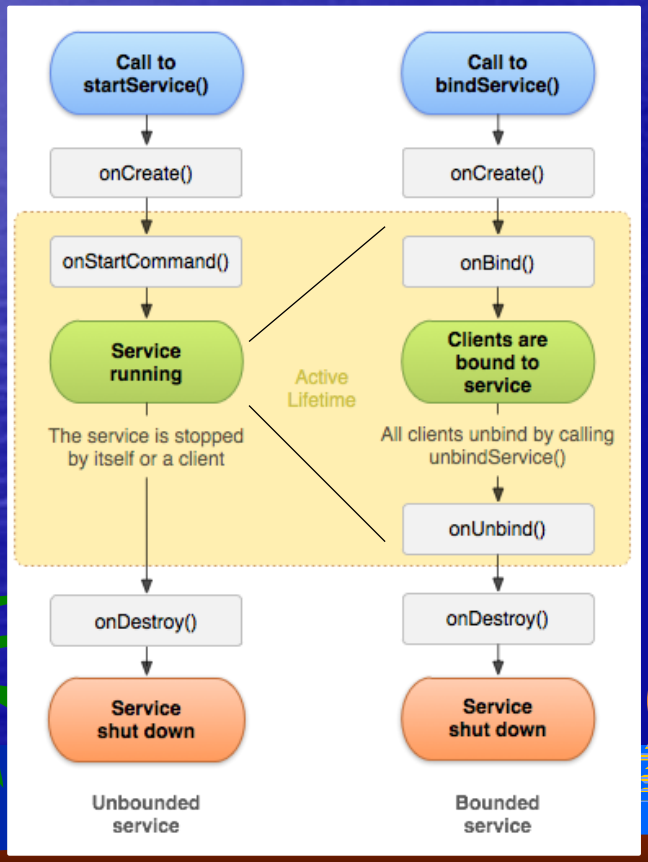
Services

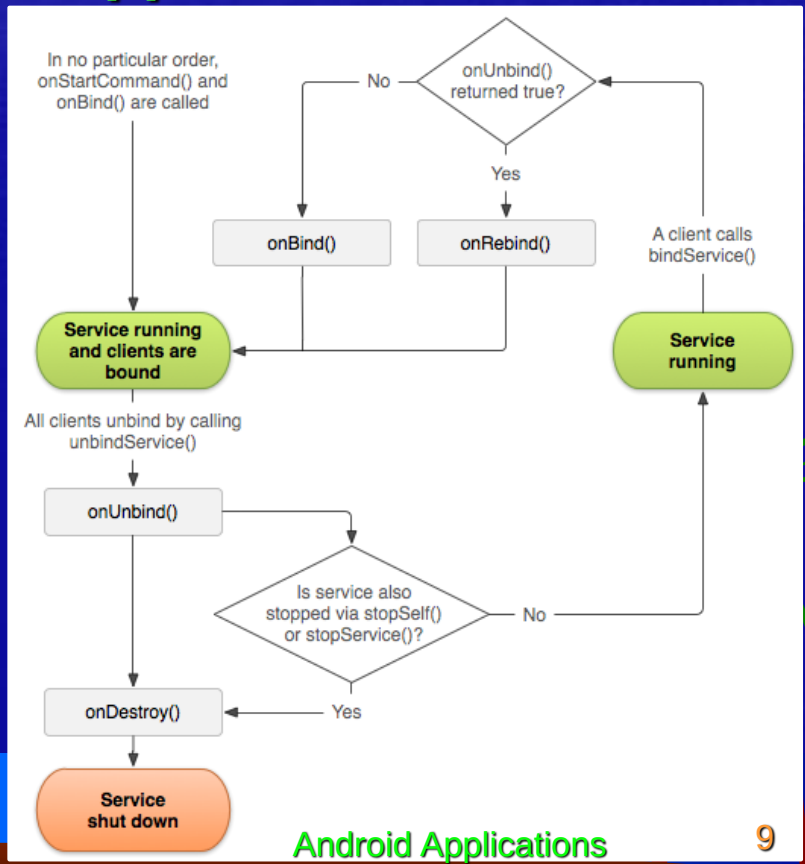
- Services are freed when
  - Stopped explicitly
    - stopService() from the client
    - stopSelf() on the service itself
  - Android needs the memory or resources they occupy, terminating the service (always after onStartCommand() had returned)
    - Services have high priorities, but less then the active Activity
- They can be automatically brought to memory again if terminated by Android
  - Depending on the onStartCommand() return value
    - START_NOT_STICKY – they are not brought to memory until a new startService() is executed
    - START_STICKY – they are brought to memory again, but with a NULL intent
    - START_REDELIVER_INTENT - they are brought to memory again with the last processed intent

Services and their lifecycle

Creation

- Can be initiated and terminated from other parts
- Or the service can be created by a connection (bind)
- A service inherits from *android.app.Service*





Service skeleton

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {
   @Override
   public void onCreate() {
      // TODO: Actions to perform when service is created.
   }

   @Override
   public IBinder onBind(Intent intent) {
      return null;   // mandatory but should return null for
                     // non remote call services
   }

   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      // Usually launch a background thread to do processing.
      return Service.START_NOT_STICKY;    // or other value
   }

   @Override
   public void onDestroy() {
      // TODO: Actions to perform when service is destroyed
   }
}
```

Manifest:

```
<service   android:name=".MyService"/>
```

Calling the service

```
// Implicitly start a Service
Intent myIntent =
          new Intent(MyService.ORDER_PIZZA);
myIntent.putExtra("TOPPING", "Margherita");
startService(myIntent);


// Explicitly start a Service in the same process
startService(new Intent(this, MyService.class));
```

Stopping the service

```
// With the same intent
stopService(new
              Intent(MyService.ORDER_PIZZA));

// Stop a service with the service name (same proc).
ComponentName service =
     startService(new Intent(this, MyService.class));
...
stopService(new Intent(this, service.getClass()));

// Stop a service explicitly in the same process
Class serviceClass =
        Class.forName(service.getClassName());
stopService(new Intent(this, serviceClass));
```

IntentService

- It's a special purpose Service subclass that creates a single worker thread
  - The intent received on onStartCommand() is passed to the method that the worker thread executes
  - Successive calls on onStartCommand() are queued
  - You only have to override and implement onHandleIntent()

```
public class MyService extends IntentService {
    public MyService() {
        super("MyService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // Do the work in this single worker thread
        // and return
    }
}
```

ResultReceiver

- Mechanism to return a result to an Activity (or other component activated by an Intent) from other component or thread (using a Handler())
  - It is created on the destination with onReceiveResult() overridden
  - As this class is Parcelable their objects can be passed in Intents
  - The recipient sends results using send(), triggering a call to onReceiveResult()

Example of ResultReceiver

```
//recipient Activity
public class MainActivity extends AppCompatActivity {
   // some variables (Activity state)
   ....

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      ...
      Intent aService = new Intent(this, MyService.class);
      aService.putExtra(MyService.RESULT, new ResultReceiver(new Handler()) {
         @Override
         protected void onReceiveResult(int code, Bundle data) {
            super.OnReceiveResult(code, data);
            ....  // if code OK, use data and other Activity state
            ....
         }
      });
   }
}
```

Recipient component
(an Activity)

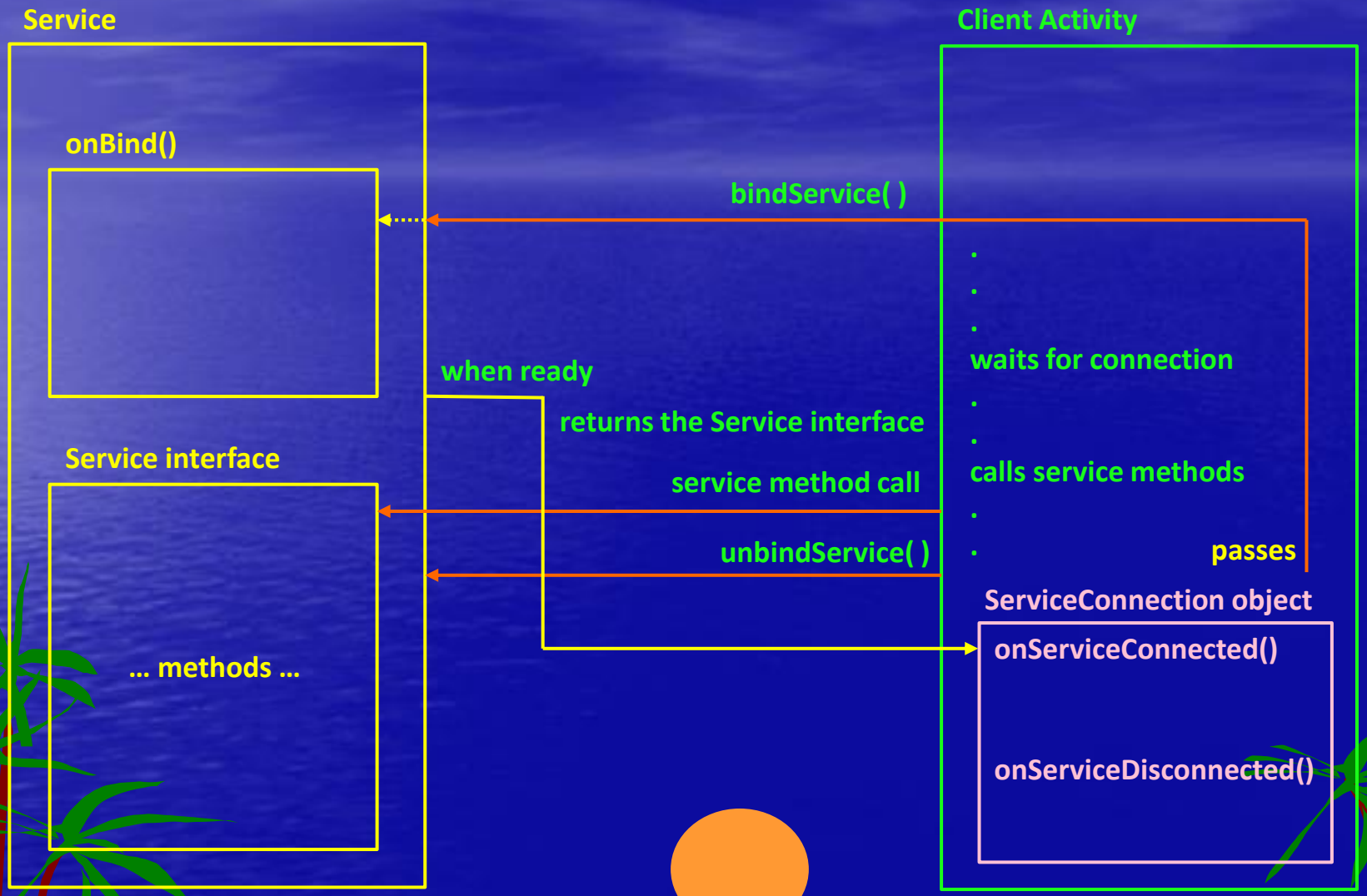
A service sending results

```
public class MyService extends Service {
   public final static String RESULT = "RemoteResult";
   ....
   @Override
   public int onStartCommand(Intent i, int flags, int sId) {
      ResultReceiver rec = i.getParcelableExtra(MyService.RESULT);
      ....
      Bundle data = new Bundle();
      data.putString("value", "some data");
      ....
      rec.send(1, data);
      return Service.START_NOT_STICKY;
   }
   ....
}
```

Remote call services

- Their functionality is invoked using RPC
  - Predefined interface specified via an AIDL file
  - Usually, they are standalone in their own processes
  - Remote call services are activated (brought to memory and onCreate() invoked) through bindService() and can be freed when the last bound client calls unbindService()
    - When a service is ready to be called through its interface a callback onServiceConnected() is called on the client
    - There is also a onServiceDisconnected() callback on the client that is called when the service is not available (motivated by a crash or reclaimed by Android)

Remote call service



Example

Service interface is defined in an AIDL file

```
// This file is IStockQuoteService.aidl
package com.androidbook.services.stockquoteservice;

interface IStockQuoteService {
   double getQuote(String ticker);
}
```

The service must implement the interface

```
public class StockQuoteService extends Service {
   public class StockQuoteServiceImpl extends
                                        IStockQuoteService.Stub {
      @Override
      public double getQuote(String ticker)
                                        throws RemoteException {
         return 20.0;
      }
   }

   @Override
   public IBinder onBind(Intent intent) {
      return new StockQuoteServiceImpl();
   }
}
```

The client calling the service

```
...
bindService(new Intent(IStockQuoteService.class.getName()),
                  serConn, Context.BIND_AUTO_CREATE);
...

private ServiceConnection serConn  =  new ServiceConnection()  {
   @Override
   public void onServiceConnected(ComponentName name,
                                        IBinder service) {
      stockService = IStockQuoteService.Stub.asInterface(service);
      callBtn.setEnabled(true);
    }
   @Override
   public void onServiceDisconnected(ComponentName name)  {
      callBtn.setEnabled(false);
      stockService = null;
   }
};

...
try {
   double val = stockService.getQuote("ANDROID");
   Toast.makeText(this, "Value from service is " + val,
                        Toast.LENGTH_SHORT)
      .show();
} catch (RemoteException ee) {
}
```

Notifications

- Are shown in the status bar
  - More details listed in the extended status drawer
  - They can produce sound, vibration and light leds
  - Created using a system service

```
String svcName = Context.NOTIFICATION_SERVICE;
NotificationManager notificationManager;
notificationManager = (NotificationManager) getSystemService(svcName);
```

  - Specified in a Notification object through a Build class

```
// A small icon, a title and a text and mandatory (many other features)
// get the Notification object using the build() method
Notification notf = new Notification.Builder(this)
    .setContentText(message)                  // the main text of the notification
    .setContentTitle(title)                   // the first line (title)
    .setSmallIcon(R.drawable.nticon)          // icon on bar and notification
    .setWhen(System.currentTimeMillis())      // for ordering
    .setPendingIntent(PendingIntent pi)       // Activity to launch on tap
    .build();                                 // returns the notification object
notf.flags |= Notification.FLAG_ONGOING_EVENT;      // cannot be cleared
```

  - Sent using the notify() method of the service

Extended Notification Drawer

Customized view notification with a RemoteViews object featuring an Icon, TextView and ProgressBar

Notifications with standard views

A customized notification

Layout specification

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:padding="5dp"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
      <ImageView android:id="@+id/status_icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true“ />
      <RelativeLayout android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:paddingLeft="10px"
        android:layout_toRightOf="@id/status_icon">
          <TextView android:id="@+id/status_text"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:textColor="#000"
            android:textSize="14sp"
            android:textStyle="bold“  />
          <ProgressBar  android:id="@+id/status_progress"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_below="@id/status_text"
            android:progressDrawable="@android:drawable/progress_horizontal"
            android:indeterminate="false"
            android:indeterminateOnly="false“ />
      </RelativeLayout>
</RelativeLayout>
```

Building the notification

```
Intent intent = new Intent(this, MyActivity.class);
PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0));
Notification notification = new Notification.Builder(this)
    .setSmallIcon(R.drawable.icon)
    .setContentText(“Custom Content”)
    .setWhen(System.currentTimeMillis())
    .setCustomContentView(new RemoteViews(this.getPackageName(),
                                          R.layout.my_status_window_layout)
    .setPendingIntent(pi);
    .build();
// allowing updates
notification.flags |= Notification.FLAG_ONGOING_EVENT;
// Putting state on the layout
notification.contentView.setImageViewResource(R.id.status_icon,
                                                R.drawable.icon);
notification.contentView.setTextViewText(R.id.status_text,
                                          "Current Progress:");
notification.contentView.setProgressBar(R.id.status_progress,
// emitting the notification                              100, 50, false);
int notificationRef = 1;
notificationManager.notify(notificationRef, notification);
```

Cancel

```
// cancelling the notification
notificationManager.cancel(notificationRef);
```

Alarms

- Calls an application component periodically or after a specified time interval
  - Uses another system service

```
String svcName = Context.ALARM_SERVICE;
AlarmManager alarms;
alarms = (AlarmManager) getSystemService(svcName);
```

  - We can use the methods set(), setRepeating() or setInexactRepeating() to create alarms

```
int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;
long timeOrLengthOfWait = 10000;
String ALARM_ACTION = "ALARM_ACTION";

Intent intentToFire = new Intent(ALARM_ACTION);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intentToFire, 0);

alarms.set(alarmType, timeOrLengthOfWait, pendingIntent);
```