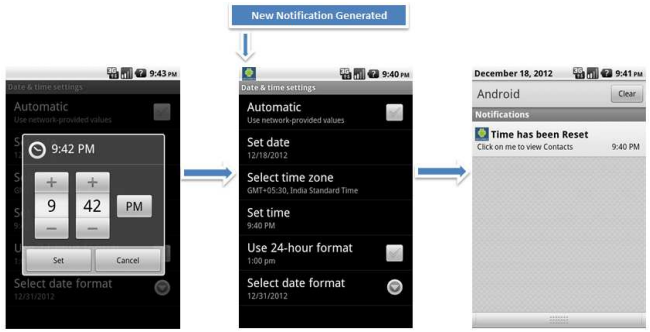
ANDROID APPS DEVELOPMENT FOR MOBILE AND TABLET DEVICE (LEVEL II)

Lecture 6: Notification and Web Services

Peter Lo

Notification

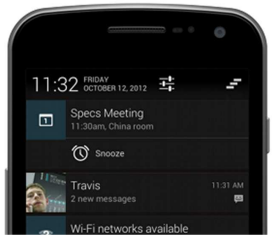
- A notification is a user interface element that you display outside your app's normal UI to indicate that an event has occurred.
- Users can choose to view the notification while using other apps and respond to it when it's convenient for them.



Notification Type

- When you tell the system to issue a notification, it first appears as an icon in the notification area.
- User opens the notification drawer to see the detail notification
- Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Notifications in the notification area.



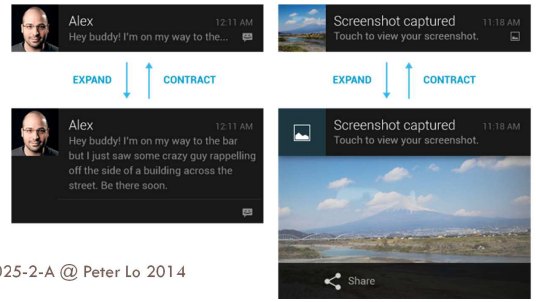
Notifications in the notification drawer

Base Layout

- All notifications consist of a base layout, including:
  - The sending application's notification icon or the sender's photo
  - A notification title and message
  - A timestamp
  - A secondary icon to identify the sending application when the senders image is shown for the main icon

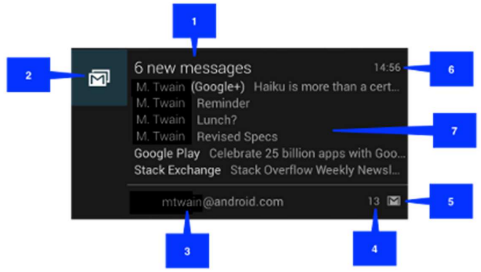
Expanded Layouts

- This provides the user with additional context, and may allow the user to read a message in its entirety.
- The user can pinch-zoom or two-finger glide in order to toggle between base and expanded layouts.
- For single event notifications, Android provides two expanded layout templates (text and image) for you to re-use in your application



Expanded Layout Elements

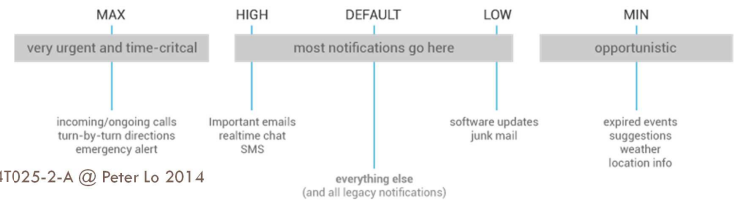
- The callouts in the illustration refer to the following:
  1. Content title
  2. Large icon
  3. Content text
  4. Content info
  5. Small icon
  6. Time that the notification was issued. You can set an explicit value with *setWhen()*; if you don't it defaults to the time that the system received the notification.



Notification Priority Management

- Android supports a priority flag for notifications.

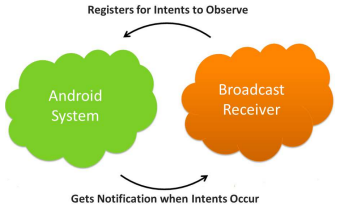
Priority	Use
MAX	Use for critical and urgent notifications that alert the user to a condition that is time-critical or needs to be resolved before they can continue with a particular task.
HIGH	Use high priority notifications primarily for important communication, such as message or chat events with content that is particularly interesting for the user.
DEFAULT	The default priority. Keep all notifications that don't fall into any of the other categories at this priority level.
LOW	Use for notifications that you still want the user to be informed about, but that rate low in urgency.
MIN	Contextual/background information (e.g. weather information, contextual location information). Minimum priority notifications will not show in the status bar. The user will only discover them when they expand the notification tray.



Create a Notification Builder

- When creating a notification, specify the UI content and actions with a *NotificationCompat.Builder* object.
- At bare minimum, a Builder object must include the following:
  - Small icon, set by *setSmallIcon()*
  - Title, set by *setContentTitle()*
  - Detail text, set *by setContentText()*

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!");
```



Define the Notification's Action

- An action takes users directly from the notification to an Activity in your application, where they can look at the event that caused the notification or do further work.
- Inside a notification, the action itself is defined by a *PendingIntent* containing an Intent that starts an Activity in your application.
- To associate the *PendingIntent* created in the previous step with a gesture, call the appropriate method of *NotificationCompat.Builder*.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
...
// Because clicking the notification opens a new ("special") activity, there's
// no need to create an artificial back stack.
PendingIntent resultPendingIntent =
    PendingIntent.getActivity(
    this,
    0,
    resultIntent,
    PendingIntent.FLAG_UPDATE_CURRENT
);
```

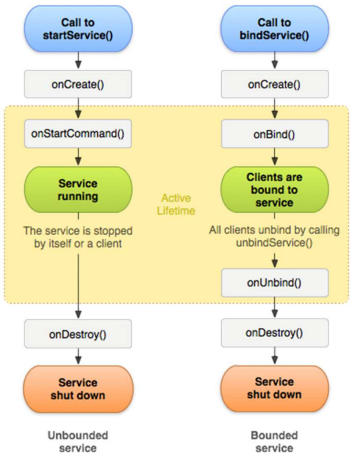

Issue the Notification

- To issue the notification:
  - Get an instance of *NotificationManager*.
  - Use the *notify()* method to issue the notification. When you call *notify()*, specify a notification ID. You can use this ID to update the notification later on.
  - Call *build()*, which returns a Notification object containing your specifications.

```
NotificationCompat.Builder mBuilder;
...
// Sets an ID for the notification
int mNotificationId = 001;
// Gets an instance of the NotificationManager service
NotificationManager mNotifyMgr =
        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
// Builds the notification and issues it.
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

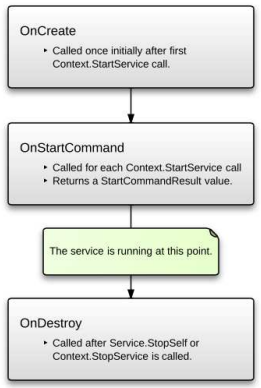

Services

- A Service is an application component that can perform long-running operations in the background and does not provide a user interface.
  - A component can start a service and it will continue to run in the background even if the user switches to another application.
  - A component can bind to a service to interact with it and even perform IPC.



Started Service

- A service is "started" when an application component (such as an activity) starts it by calling *startService()*.
- Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
- Usually, a started service performs a single operation and does not return a result to the caller.
  - For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.



Bound Services

- A service is "bound" when an application component binds to it by calling *bindService()*.
- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with Inter-Process Communication (IPC).
- A bound service runs only as long as another application component is bound to it.
- Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Create an IntentService

- To create an *IntentService* component for your app, define a class that extends *IntentService*, and within it, define a method that overrides *onHandleIntent()*.

```
public class RSSPullService extends IntentService {
    @Override
    protected void onHandleIntent(Intent workIntent) {
        // Gets data from the incoming Intent
        String dataString = workIntent.getDataString();
        ...
        // Do work here, based on the contents of dataString
        ...
    }
}
```


Define the IntentService in the Manifest

- An *IntentService* also needs an entry in your application manifest. Provide this entry as a <service> element that's a child of the <application> element:

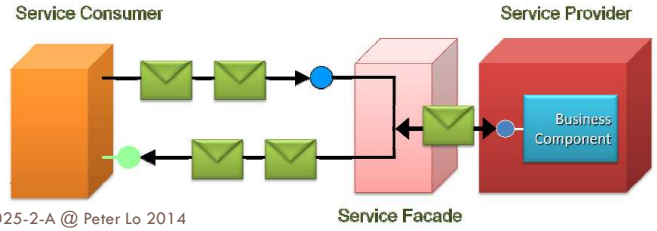
```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    ...
    <!--
        Because android:exported is set to "false",
        the service is only available to this app.
    -->
    <service
        android:name=".RSSPullService"
        android:exported="false"/>
    ...
<application/>
```


Thread vs. Service & Asynchronous Task

- **Service** is like an Activity but has no interface. Probably if you want to fetch the weather for example you won't create a blank activity for it, for this you will use a Service.
- A **Thread** is a Thread, probably you already know it from other part. You need to know that you cannot update UI from a Thread. You need to use a Handler for this, but read further.
- An **AsyncTask** is an intelligent Thread that is advised to be used. Intelligent as it can help with it's methods, and there are two methods that run on UI thread, which is good to update UI components

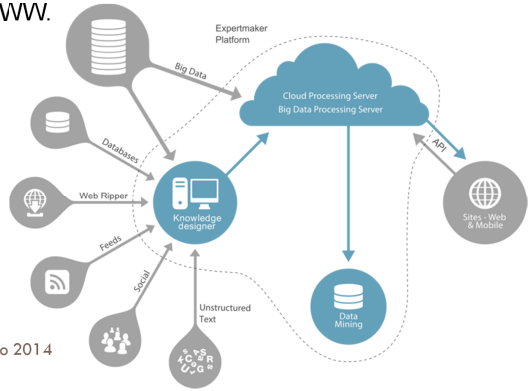
Service Oriented Architecture (SOA)

- SOA is a design approach for building business applications as a set of loosely coupled black box components orchestrated to deliver a well-defined level of service by linking together business processes.
- The SOA approach is the best way to achieve business agility which is the ability to change the business process quickly in response to the change in the business environment, such as adding a new service to the organization portfolio.



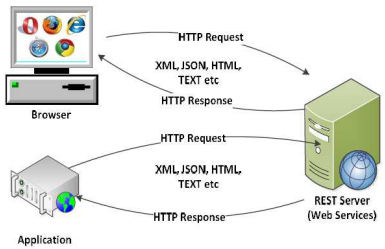
Web Service

- Web services are software systems that have taken the concept of services delivered over the web to support interoperable machine-to-machine interaction over a network
- It is a method of communications between two electronic devices over the WWW.



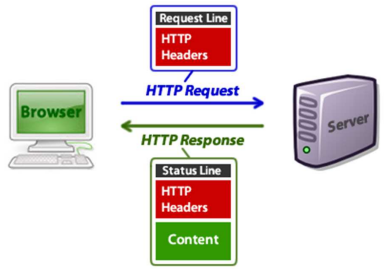
RESTful Web services

- REST (Representational State Transfer) is a technology relies on a stateless, client-server, cacheable communications technology that uses the HTTP protocol
- In REST, the web services are viewed as resources and can be identified by their URLs.
- Web service clients that want to use these resources and access a particular representation will need to use a globally defined set of remote methods that describe the action to be performed on the resource



HTTP GET

- The GET method means retrieve whatever information is identified by the Request-URI.
- If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.



Consuming HTTP GET Requests

- Android applications pull information from various sources
- A common integration strategy is to use HTTP
- In order to consuming HTTP GET request:
  - Create an *HttpClient*.
  - Instantiate a new HTTP method, such as *PostMethod* or *GetMethod* .
  - Set HTTP parameter names/values.
  - Execute the HTTP call using the *HttpClient*.
  - Process the HTTP response.

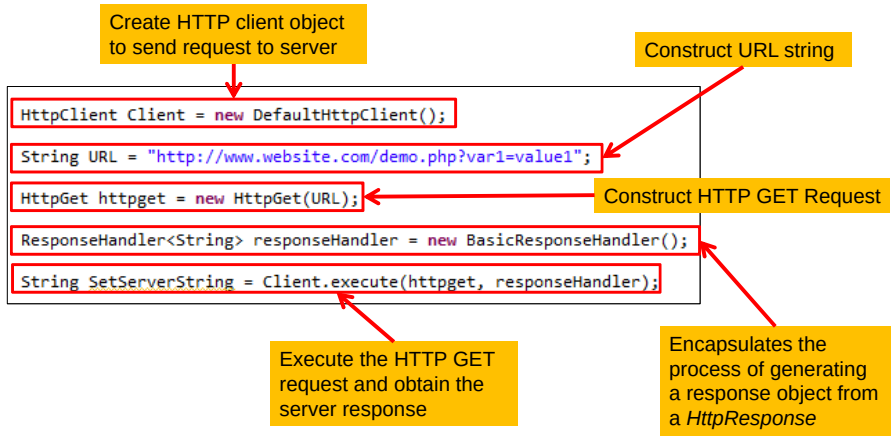
HTTP Operations

- Based on Apache HTTP package
- HttpClient interface
  - Interface for an HTTP client
  - HTTP clients encapsulate a smorgasbord of objects required to execute HTTP requests while handling cookies, authentication, connection management, and other features
  - Thread safety of HTTP clients depends on the implementation and configuration of the specific client
- DefaultHttpClient class
  - Default implementation of HttpClient interface

Receiving HTTP Response

- HttpResponse
  - Deals with responses in HTTP MIME type
- RespondHandler interface
  - Handler that encapsulates the process of generating a response object from a HttpResponse
- BasicResponseHandler
  - A ResponseHandler that returns the response body as a String for successful (2xx) responses. If the response code was >= 300, the response body is consumed and an HttpResponseException is thrown.

Create a HTTP GET Request



Addressing Multithreading Issues

- Create one *HttpClient* for the entire application and all HTTP communication
- Pay attention to multithreading issues when making simultaneous requests through the same *HttpClient*
- *HttpClient* provides facilities that make this easy – create the *DefaultHttpClient* using a *ThreadSafeClientConnManager*.

JSON

- JSON (JavaScript Object Notation) is an independent data exchange format.
- JSON is limited to text and numeric values.
  - Binary values are not supported.
- JSON is a subset of the JavaScript Specification (ECME-Script) and it is therefore directly supported in JavaScript.
- Data structures in JSON are based on key / value pairs.
  - The key is a string, the value can be a numerical value, a Boolean value (true or false) or an object.

JSON Structure

- JSON is built on two structures:
  - **JSON Object:** A collection of name/value pairs which represented by { }.

```
{
   "employee": {
      "name":     "sachin",
      "salary":   56000,
      "married":  true
   }
}
```

  - **JSON Array:** An ordered list of values which represented by [ ].

```
{ "Employee" :
   [
    {"id":"101","name":"Sonoo Jaiswal","salary":"50000"},
    {"id":"102","name":"Vimal Jaiswal","salary":"60000"}
   ]
}
```


JSON String

- Consider this sample JSON string:

```
{"FirstObject": { "attr1":"one value" ,"attr2":"two value",
"sub": { "sub1":[ {"sub1_attr":"sub1_attr_value" },{"sub1_attr":"sub2_attr_value" }]}
}
}
```

- Or we can reformat it as follow:

```
{
    "FirstObject": {
        "attr1": "one value",
        "attr2": "two value",
        "sub": {
            "sub1": [
                {
                    "sub1_attr": "sub1_attr_value"
                },
                {
                    "sub1_attr": "sub2_attr_value"
                }
            ]
        }
    }
}
```

Parsing JSON Object