ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2020 - Materia: **Aplicaciones Android**

TEMA: App & Activities

Consideraciones

- Estas transparencias **no** tienen el objetivo de suplir las clases.
- Por tanto, serán **complementadas** con ejemplos, códigos, profundizaciones y comentarios por parte del docente.
- El **orden** de dictado de estos temas está sujeto a la consideración del docente.

Referencias

- Documentación para developers:
  - *http://developer.android.com/*
- Aspectos fundamentales de la aplicación:
  - *https://developer.android.com/guide/components/fundamentals.html*
- Manifiesto de la app:
  - *https://developer.android.com/guide/topics/manifest/manifest-intro*
- Provisión de recursos:
  - *https://developer.android.com/guide/topics/resources/providing-resources*
- Acceso a recursos:
  - *https://developer.android.com/guide/topics/resources/accessing-resources*

Agenda

- Entorno de Ejecución
- Anatomía de una Aplicación
- Manifiesto
- Recursos
- Tipos de Componentes
- Activities - Creación
- Activities - Ciclo de Vida

Entorno de Ejecución (1)

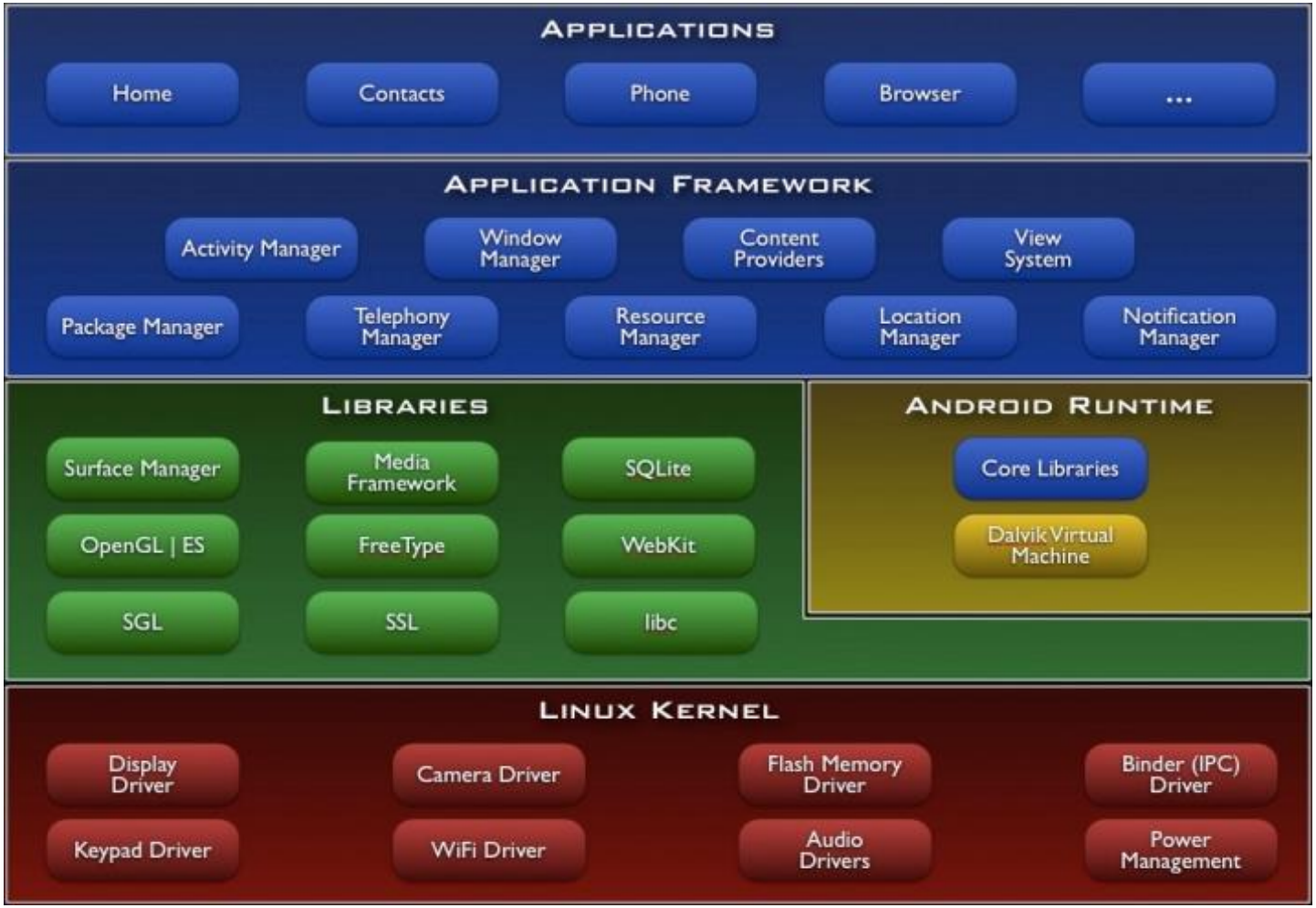
- Las aplicaciones Android son escritas en lenguaje Java
  - IDE (Eclipse/Android Studio) + JDK + Android SDK
- El SDK de Android compila código fuente y empaqueta recursos requeridos en un archivo de tipo APK (Android Package)
- Se crea una Maquina Virtual (VM) llamada Dalvik, para cada aplicación
- **Seguridad**:
  - Android utiliza el principio de "least privilege" (mínimos privilegios). Dado el nivel de granularidad en seguridad, es necesario "pedir" los permisos explícitamente cuando se crea una aplicación.

Entorno de Ejecución (2)

- **Seguridad (cont.)**:
  - Una vez instalada, la aplicación se ejecuta en su propio contexto de seguridad.
    - Android esta basado en el SO multi-usuario Linux.
    - Cada aplicación es tratada como un usuario diferente (se le asigna un user ID de Linux único).
    - Permisos asociados al ID de usuario de una aplicación para todos los archivos.
    - Cada aplicación se ejecuta en su propia VM de Android y en su propio proceso Linux (asegura aislamiento de otras aplicaciones).

Entorno de Ejecución (3)

- **Arquitectura:**



Anatomía de una Aplicación (1)

- **Código fuente**

```
public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

- **Manifiesto**
  - Información requerida para iniciar la aplicación Android.
  - Componentes, permisos, funcionalidad de HW y SW
  - a utilizar, API Level, etc.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example" android:installLocation="internalOnly" >

    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="21" />

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

    <application
        android:name="com.example.Application"
        android:allowBackup="false"
        android:allowClearUserData="true"
        android:icon="@drawable/ic_launcher"
        android:label="Qoollet"
        android:logo="@drawable/_q"
```

Anatomía de una Aplicación (2)

- **Recursos**
  - Recursos adicionales utilizados por código fuente
  - Ejemplo: Imágenes, UI Layouts, cadenas de texto para internacionalización, etc.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="ma
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView android:text="Hello world!" android:layout_width="wrap_conten
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Manifiesto (1)

- Tiene como primer objetivo permitir al sistema Android identificar propiedades del componente a ejecutar.
- Siempre ubicado en el directorio raíz del proyecto.
- Se declaran los permisos y funcionalidades del dispositivo que la aplicación necesita para ejecutarse.
  - Permiso para acceder a Internet:

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

  - Permiso para acceder a contactos:

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

  - Solicitar uso de funcionalidad “Camara” (front/rear):

```
<uses-feature android:name="android.hardware.camera.any”
        android:required="true" />
```

Manifiesto (2)

- Se declaran todas las actividades y componentes, y las interacciones que estos manejan / atienden.
- Nombre, icono, logo de la aplicación.
- Meta-información para ser utilizada desde la aplicación empaquetada.
  - Información de versiones o librerías.
  - Tokens de seguridad, expiración de licencias.

Recursos

- Se desacoplan los recursos estáticos del código Java.
  - Imágenes, audio, etiquetas de texto, temas/estilos.
- Relacionado al problema de multi-dispositivos, multi-resolución, multi-lenguaje.
  - Se incluyen en una misma aplicación, recursos diferentes para diferentes configuraciones de dispositivos.
- Mediante convención de nombres, se permite utilizar los recursos desde el codigo Java.

Tipos de componentes (1)

- **Activities**
  - Representan las pantallas mediante las cuales el usuario final interactúa con la aplicación.
  - Para desarrollar estas pantallas se debe extender de la clase ***Activity***.
- **Services**
  - Componentes que se ejecutan sin necesidad de una interfaz de usuario.
  - Ejemplos: tareas que a priori no se sepa su duración, interacción con otros sistemas remotos que puedan generar demoras y/o errores, etc.
  - Para crearlos se debe extender de la clase ***Service***.

Tipos de componentes (2)

- **Content providers**
  - Manejan el acceso a repositorios centrales de contenido desde otras aplicaciones.
  - Usualmente proveen de una interfaz de usuario para su propósito.
  - Para crearlos se debe extender de la clase ***ContentProvider***, definir permisos de acceso, UI, etc.
- **Broadcast receivers**
  - Componentes que responden a mensajes emitidos hacia todo el sistema Android.
  - Pueden ser emitidos por el sistema (batería baja, foto capturada, estado de conexión, etc.) o por aplicaciones.
  - Para desarrollarlos se debe extender de la clase ***BroadcastReceiver***.

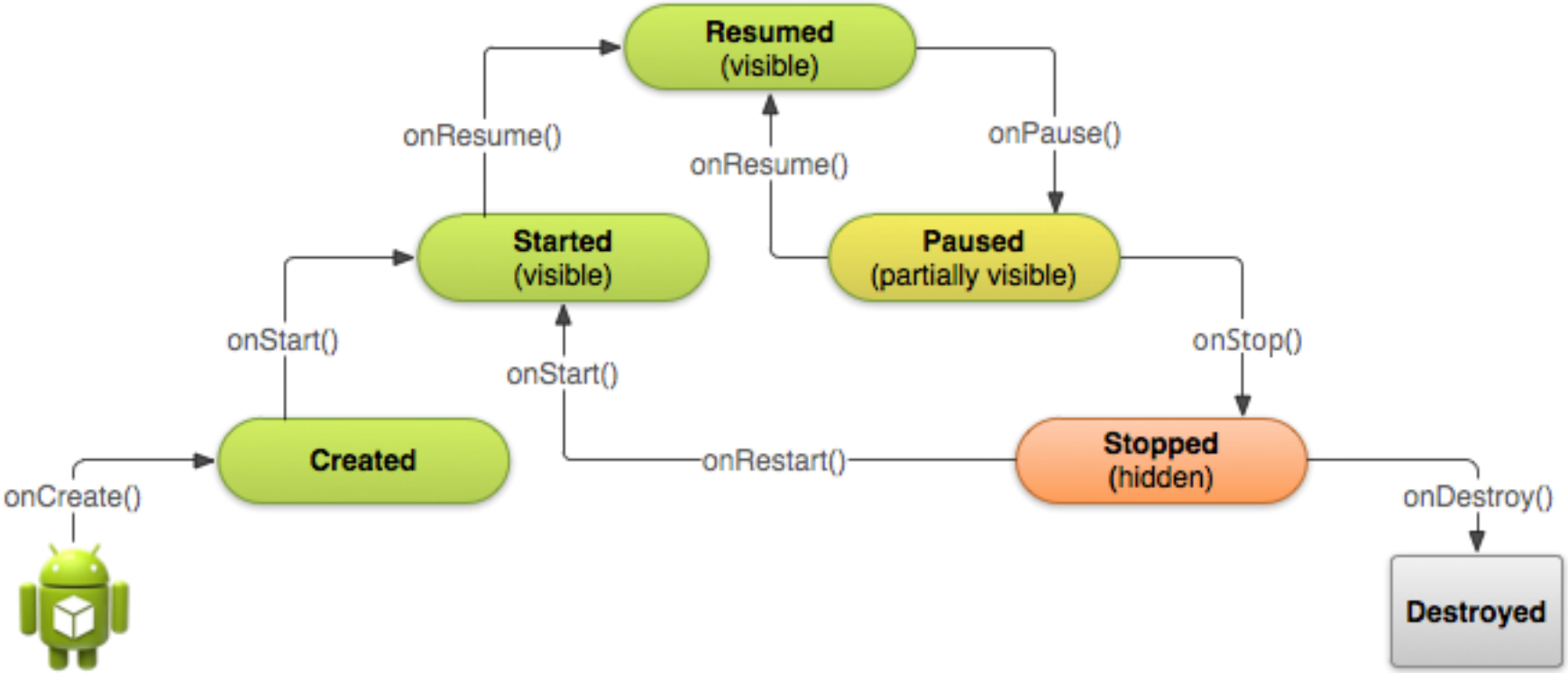
Activities: Creación (1)

- Se debe crear una clase que extienda de la clase Activity, o de alguna subclase de esta con funcionalidad ya provista (Ej: AppCompatActivity)
- Implementar mínimamente los métodos:
  - **onCreate**(): es obligatorio reimplementar este método. Aquí es donde debe incluirse todo lo necesario para que la actividad funcione correctamente. Adicionalmente, es donde se le debe indicar a la plataforma Android el archivo de recurso con el diseño de la interfaz a utilizar.
  - **onPause**(): significa que el usuario esta yéndose de esta actividad, y deberíamos tomar todas las acciones para dejar nuestra aplicación consistente (el usuario podría volver o no).

Activities: Creación (2)

- Se debe declarar en el manifiesto la existencia de la nueva actividad:
  - manifest -> application -> activity
  - Atributos principales: nombre, etiqueta, icono, tema
- Para que la actividad pueda ser creada, es necesario agregar intent-filters que le dicen a la plataforma en que condiciones se ejecuta la actividad, donde se muestra disponible en el sistema para ser iniciada por el usuario, etc.
- ***action.MAIN*** especifica la actividad a iniciar cuando la aplicación es ejecutada por primera vez
- ***category.LAUNCHER*** ofrece al usuario la actividad en la lista de aplicaciones

Activities: Ciclo de Vida (1)



Activities: Ciclo de Vida (2)

- **Resumed** = visible al usuario
- **Stopped** = oculta al usuario, estado aún existente
- **Paused** = la actividad esta "obscurecida" y tiene otra encima (no recibe interacciones del usuario)
- Todos los métodos del ciclo son invocados por el sistema.
- Dependiendo de la complejidad de la aplicación, no se necesitan sobrescribir todos los métodos del ciclo.
- Beneficios de implementarlo correctamente:
  - Permitir que el usuario reciba una llamada mientras utiliza la aplicación.
  - No consumir recursos cuando la aplicación no esta siendo utilizada.
  - Mantener información de progreso o estado de uso.

Activities: Ciclo de Vida (3)

- **onPause**() → Técnicamente el usuario esta abandonando la actividad. Tareas usuales:
  - Parar acciones que consuman CPU.
  - Guardar cambios que el usuario espera tener en caso de volver a la actividad.
  - Liberar recursos compartidos y de hardware.
- **onResume**() → Se llama cuando la actividad "vuelve a la vida", incluso la primera vez que se muestra. Tareas usuales:
  - Inicializar componentes liberados durante onPause().
- **onStart() / onRestart()** → Cuando se visualiza nuevamente luego de *onStop*(), se llama a *onRestart*() y luego a *onStart*(), en ese orden. No hay una regla general para incluir comportamiento en *onRestart*(). Depende de la lógica de la app.

Activities: Ciclo de Vida (4)

- **onStop**() → La actividad ya no esta visible y se deben liberar todos los recursos. Luego de parada la actividad, Android podría liberar la memoria asignada a la misma para otro propósito.
  - Si bien *onPause*() es llamado previamente, aquí es donde se deben realizar liberaciones que consuman mas tiempo, CPU, etc.
  - Los objetos gráficos y su estado (pe: valores para EditText) son retenidos por Android por lo cual no deben ser persistidos programáticamente.
- **onDestroy**() → Último evento en el que se tendrá oportunidad de liberar recursos y dejar a la aplicación en estado consistente.
  - Reintentos de liberación no ocurridos durante *onStop*()