Tema 4:

4.1.- Más sobre Eventos:

- Escuchadores VS Manejadores
- Eventos de teclado y Pantalla Táctil
- Incluir estas posibilidades en el juego BiciDidáctico

4.2.-Sensores (NO)

4.3.-Ciclo de Vida de las Actividades Android

4.4.- Multimedia:

- Clases e interfaces para la gestion de video
- Clases e interfaces para la gestion de audio

Eventos de Teclado

onClick()	Método de la interfaz View.onClickListener. Se llama cuando el usuario selecciona un elemento.
onLongClick()	Método de la interfaz View.onLongClickListener. Se llama cuando el usuario selecciona un elemento durante más de un segundo.
onFocusChange()	Método de la interfaz View.onFocusChangeListener. Se llama cuando el usuario navega dentro o fuera de un elemento.
onKey()	Método de la interfaz View.onKeyListener. Se llama cuando se pulsa o se suelta una tecla del dispositivo.
onTouch()	Método de la interfaz View.onTouchListener. Se llama cuando se pulsa o se suelta o se desplaza en la pantalla táctil.
onCreateContextMenu()	Método de la interfaz View.onCreateContextMenuListener. Se llama cuando se crea un menú de contexto.

Manejadores de Teclado

En una Clase que herede de la clase **View** se pueden usar los siguientes manejadores de evento

onKeyDown(int keyCode, KeyEvent e)	Llamado cuando una tecla es pulsada.
onKeyUp(int keyCode, keyEvent e)	Llamado cuando una tecla deja de ser pulsada.
onTrackballEvent(MotionEvent me)	Llamado cuando se mueve el trackball.
onTouchEvent(MotionEvent me)	Llamado cuando se utilice la pantalla táctil.
onFocusChanged(boolean obtengoFoco, int direccion, Rect prevRectanguloFoco)	Llamado cuando cambia el foco.

Pantalla táctil

La `pantalla tactil podrá manejarse con la interfaz **onTouchListener** en las clases generales o con el método **onTouchEvent** en la clases VIEW.
Los método más interesantes sobre onTouchEvent son:

getAction()	Tipo de acción realizada: ACTION_DOWN, ACTION_MOVE, ACTION_UP o ACTION_CANCEL.
getX(), getY()	Posición de la pulsación.
getDownTime()	Tiempo en ms en que el usuario presionó por primera vez en una cadena de eventos de posición.
getEventTime()	Tiempo en ms del evento actual.
getPression()	Estima la presión de la pulsación. El valor 0 es el mínimo, el valor 1 una pulsación normal.
getSize()	Valor entre 0 y 1 que estima el grosor de la pulsación.

Modificaciones en VistaJuego

```
public boolean onKeyDown(int codigoTecla, KeyEvent evento) {
        super.onKeyDown(codigoTecla, evento);
        boolean pulsacion=true;   //Procesamos la pulsación
        switch (codigoTecla) {
        case KeyEvent.KEYCODE_DPAD_UP:
                aceleracionBici=+PASO_ACELERACION_BICI;break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
                giroBici=-PASO_GIRO_BICI; break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
                groBici=+PASO_GIRO_BICI;  break;
        case KeyEvent.KEYCODE_DPAD_CENTER:
        case KeyEvent.KEYCODE_ENTER:
                lanzarRueda();break;
        default:
                //Si estamos aquí no hemos pulsado nada que  interese
                pulsacion=false; break;
                }
                return pulsacion;
        }
```

Modificaciones en VistaJuego

```
public boolean onTouchEvent(MotionEvent evento) {
   super.onTouchEvent(evento);
   //Obtenemos la posición de la pulsación
  float x=evento.getX(); float y=evento.getY();
  switch (evento.getAction()) {
        case MotionEvent.ACTION_DOWN:// se activa la variable disparo
                disparo=true;
                break;
        case MotionEvent.ACTION_MOVE:
                float dx=Math.abs(x-mX);float dy=Math.abs(y-mY);
                //Un desplazamiento  horizontal hace girar la bici.
                if (dy<6 && dx>6)
                {giroBici = Math.round((x-mX)/2);
                  disparo = false;
                } else //desplazamiento vertical produce aceleración.
                  if (dx<6 && dy>6)
                        {aceleracionBici = Math.round((mY-y)/25);
                        disparo = false;
                }
                break;
        case MotionEvent.ACTION_UP://levatar dedo sin despazar =>disparo
                giroroBici = 0;aceleracionBici = 0;
                if (disparo) lanzarRueda();
                break;

    }
    mX=x; mY=y;
    return true;
   }
```

Ciclo de Vida: Actividades Android

- Una aplicación en Android va a estar formada por un conjunto de elementos de visualización: Actividades.
- Además de varias actividades una aplicación también puede contener servicios.
- Son las actividades las que realmente controlan el ciclo de vida de las aplicaciones, dado que el usuario no cambia de aplicación, sino de actividad.
- El sistema va a mantener una pila con las actividades previamente visualizadas, el usuario puede regresar a la actividad anterior pulsando la tecla "atrás".
- Una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado para la aplicación y continuará vivo hasta el sistema reclame su memoria para asignársela a otra aplicación.

Ciclo de Vida: Actividades Android

- Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlado directamente por la aplicación. Es el sistema quien determina cuando destruir el proceso.

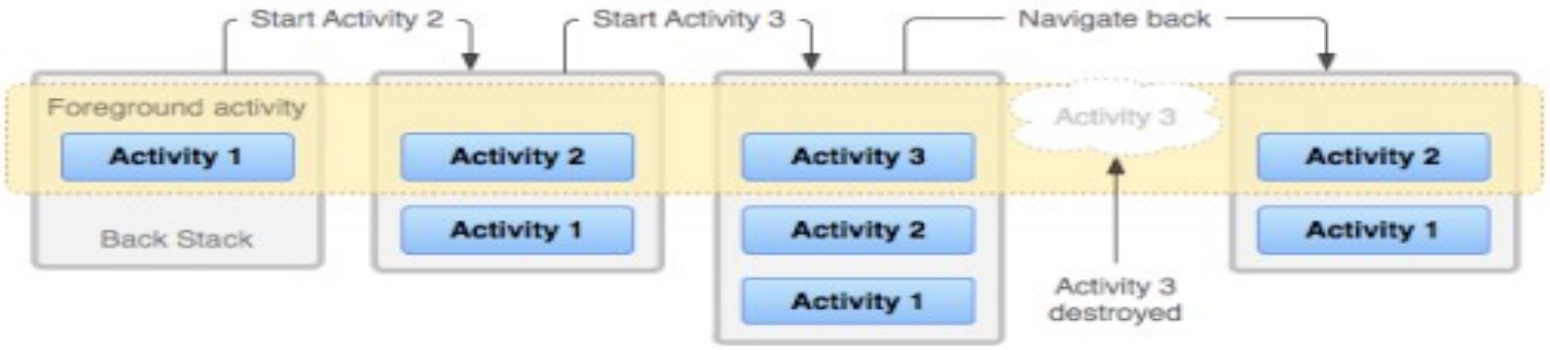
- Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenia esta aplicación.

- Es responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sea reiniciada conserve su estado.

Ciclo de Vida: Actividades Android

• Se puede lanzar nuevas actividades desde otras actividades, de tal forma que la actividad lanzadora es pausada, pero el sistema la mantiene en memoria en una cola denominada back stack.

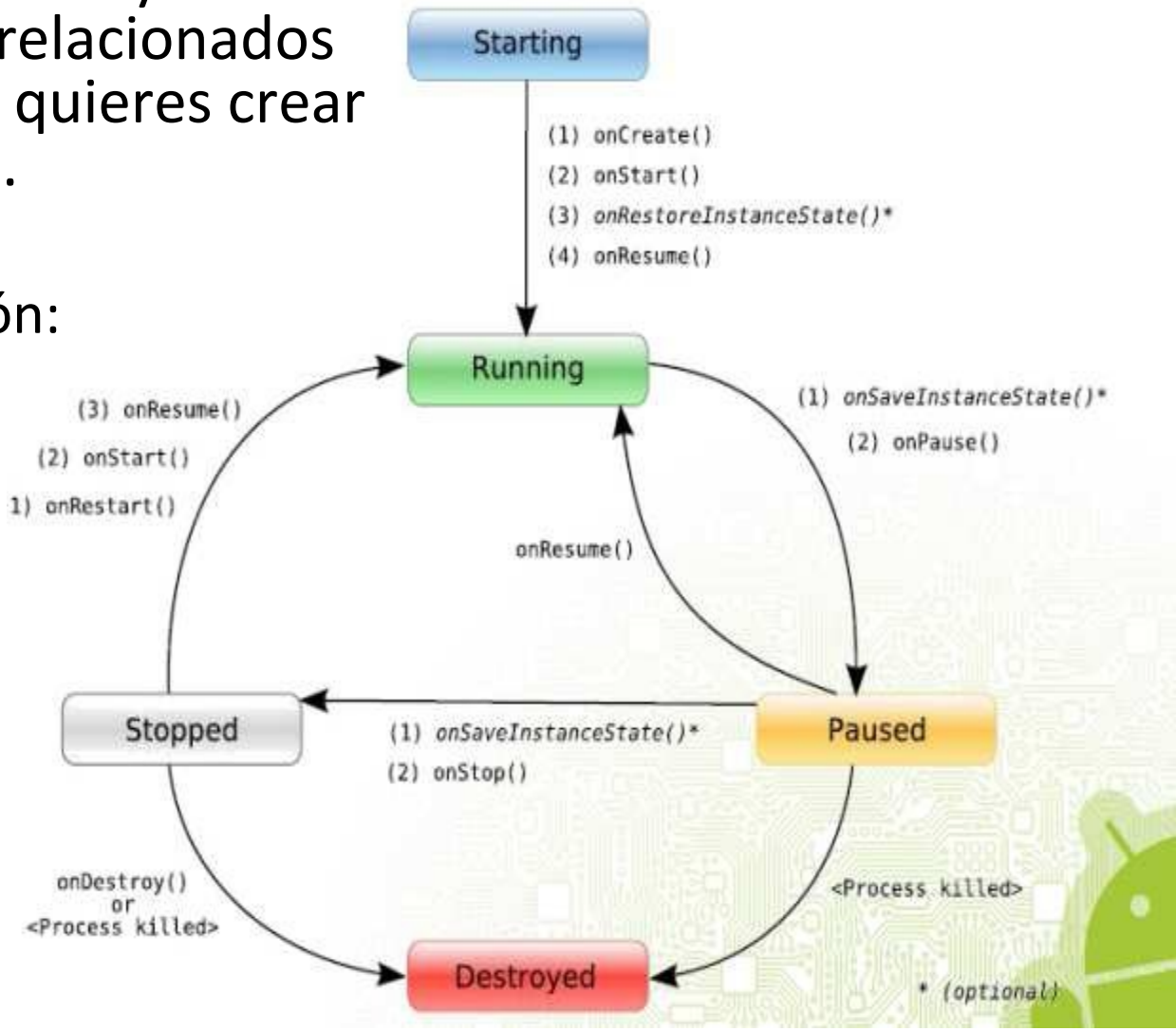
• Esta cola consiste en una cola tipo LIFO (Last In, First Out), o lo que es lo mismo, la última actividad que fue añadida, será la primera en la cola. Así, cuando el usuario pulse el botón atrás (Back), el sistema nos quitará la actividad actual y nos mostrará justo la anterior en la cola, aunque este comportamiento por defecto puede ser modificado según nuestro interés.



Estados de un Aplicación Android I

- Es necesario comprender y manejar los eventos relacionados con el ciclo de vida si quieres crear aplicaciones estables.

- Estados de un aplicación:
  - Activa
  - Visible (en pausa)
  - Parada
  - Destruida:



Estados de un Aplicación Android II

- Estados de un aplicación:

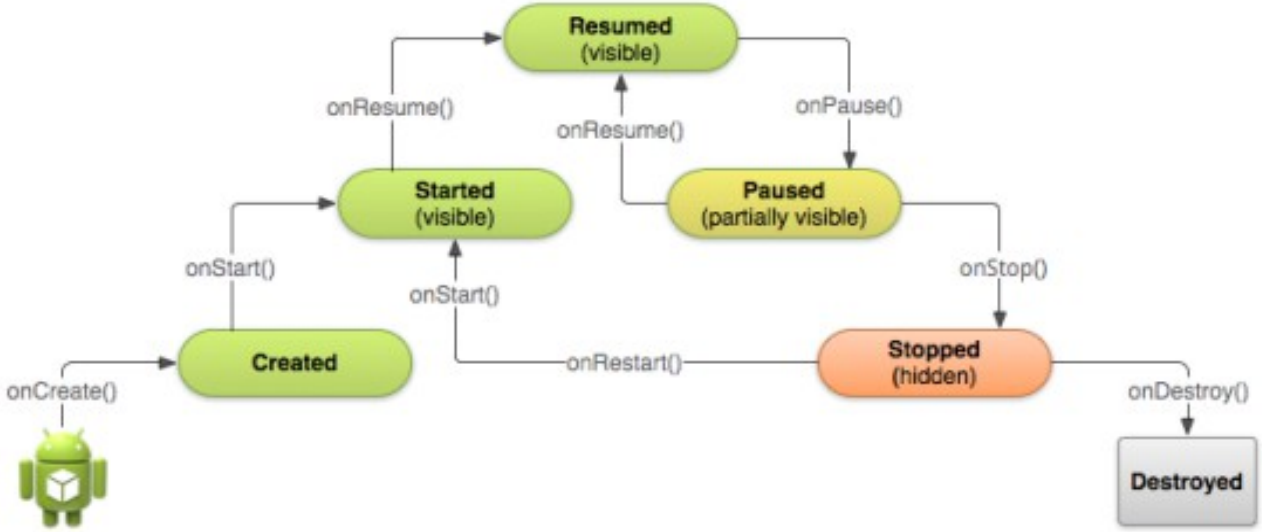
  Activa (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.

  - **Visible** (*Paused*): La actividad es visible pero SIN el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
  - **Parada** (*Stopped*): Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.
  - **Destruida** (*Destroyed*): La actividad termina al invocarse el método *finish()*, o es matada por el sistema Android, sale de la pila de actividades.

Estados de un Aplicación Android II

- Las actividades pueden ser instanciadas más de una vez. Para crear una actividad, basta con que creemos una clase que herede de la clase Activity. Además de heredar de esta clase, deberemos sobreescribir algunos métodos que pertenecen al ciclo de vida de la actividad.

- Este ciclo de vida consiste en los diferentes estados por los que puede pasar una actividad y los métodos que nos permiten cambiar de un estado a otro.



Métodos

- **onCreate(Bundle)**: Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de instancia (en una instancia de la clase Bundle), si se reanuda desde una actividad que ha sido destruida y vuelta a crear.
- **onStart()**: Nos indica que la actividad está a punto de ser mostrada al usuario.
- **onResume()**: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- **onPause()**: Indica que la actividad está a punto de ser lanzada a segundo plano, porque otra aplicación es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- **onStop()**: La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
- **onRestart()**: Indica que la actividad va a volver a ser representada después de haber pasado por *onStop()*.
- **onDestroy()**: Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón <volver> o cuando se llama al método *finish()*. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Ejemplo en SoloBici

- Introduce todos estos métodos en la clase “Solobici.java”
- Observa la secuencia de mensajes Toast

```
@Override protected void onStart() {
  super.onStart();
  Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
}
@Override protected void onResume() {
  super.onResume();
  Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
}
 @Override protected void onPause() {
  Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
  super.onPause();
}
 @Override protected void onStop() {
  super.onStop();
  Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
 }
```

Ejemplo en SoloBici

```
@Override protected void onRestart() {
  super.onRestart();
  Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
}

@Override protected void onDestroy() {
  Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
  super.onDestroy();
}
```

- Pulsa el botón *Jugar* y luego regresa a la actividad. Observa la secuencia de Toast.
- Pulsa el botón *Acerca de* y luego regresa a la actividad. Observa la secuencia de Toast.
- Sal de la actividad y observa la secuencia de Toast.

Ejemplo en SoloBici

```
class HiloJuego extends Thread {
   private boolean pausa,corriendo;
    public synchronized void pausar() {
          pausa = true;
   }
    public synchronized void reanudar() {
          pausa = false;
          notify();
   }
    public void detener() {
          corriendo = false;
          if (pausa) reanudar();
   }
```

Ejemplo en SoloBici

…….continua HiloJuego extends thread …… {

```
    @Override    public void run() {
      corriendo = true;
      while (corriendo) {
          actualizaMovimiento();
          synchronized (this) {
              while (pausa) {
                  try {
                      wait();
                  } catch (Exception e) {
                  }
              }
          }
      } // del while
  } //del metodo run
} de la clase HiloJuego
```

Ejemplo en SoloBici

3. Incluye la siguiente variable en la actividad **Juego.**

```
private VistaJuego vistaJuego;
```

4. Al final de onCreate añade:

```
vistaJuego = (VistaJuego) findViewById(R.id.VistaJuego);
```

5. Incorpora los siguientes métodos a la actividad Juego

```
@Override protected void onDestroy() {
  vistaJuego.getHilo().detener();
  super.onDestroy();

@Override protected void onPause() {
  super.onPause();
  vistaJuego.getHilo().pausar();
}
 @Override protected void onResume() {
  super.onResume();
  vistaJuego.getHilo().reanudar();
}
```

Bibliografia:

- http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-aplicacion
- http://www.jtech.ua.es/cursos/apuntes/moviles/daa2013/index.html
- http://www.elandroidelibre.com/2014/11/El%20Androide%20Libre_files/aprende-android-en-20-conceptos-conceptos-3-y-4.htm

Notacion extra

• Varias actividades pertenecerán a una tarea (task), la cual se define como un conjunto de actividades destinados a un trabajo determinado.

• A nivel de Manifest podemos gestionas las tareas, con la definición de algunos atributos (taskAffinity, launchMode, allowTaskReparenting, clearTaskOnLaunch, alwaysRetainTaskState, finishOnTaskLaunch) y flags o banderas (FLAG_ACTIVITY_NEW_TASK, FLAG_ACTIVITY_CLEAR_TOP, FLAG_ACTIVITY_SINGLE_TOP),

• El comportamiento por defecto se puede entender en este ejemplo:

- La Actividad A lanza B
- A para y guarda su estado
- Le damos el botón Home
- Se mantiene el estado de cada actividad en la tarea
- Le damos a Back
- La actividad actual de la sale de la pila backstack y se destruye