**Manejo de Threads en Android y Conceptos Básicos de Handlers y Runnables**

En Android, existen los procesos o ***threads***, los cuales son una instancia de una aplicación en ejecución. Son administrados por el SO y pueden contener una o varias actividades.

Un hilo es la unidad más pequeña de ejecución dentro de un proceso. Un proceso puede tener varios hilos que se ejecutan de manera concurrente. Estos comparten la misma memoria y recursos dentro de un proceso. Sin embargo, cada hilo tiene su propia pila de ejecución.

Es importante manejar los hilos de manera adecuada para evitar bloquear la interfaz de usuario y proporcionar una experiencia fluida a los usuarios.

Hay dos tipos principales de hilos: el hilo principal (UI thread), el cual se encarga de la interfaz de usuario, y los hilos secundarios (background threads) que se ocupan de otras operaciones en segundo plano.

*En Java se define un Thread y se inicia su ejecución:*

*Thread miThread = new Thread();*

*miThread.start();*

Se puede especificar el código a ejecutar creando una subclase de Thread o implementando la interfaz Runnable:

java

Copy code

public class MyThread extends Thread {

public void run(){

System.out.println("MyThread running");

}

}

Runnable myRunnable = new Runnable(){

public void run(){

System.out.println("Runnable running");

}

}

Thread thread = new Thread(myRunnable);

thread.start();myRunnable);

**Ciclo de Vida de un Thread en Android:**

thread.start();

Aquí se detallan los aspectos clave de este ciclo de vida:

Operaciones en un Worker-Thread:

* En Android, se realizan operaciones intensivas o de larga duración en un *worker-thread* para evitar bloquear el hilo de interfaz de usuario (UI).
* Los *worker-threads* permiten ejecutar tareas que no deben interferir con la capacidad de respuesta del UI Thread, como operaciones de red, cálculos intensivos, etc.
* Se crean instancias de Thread o se implementa la interfaz Runnable para representar las operaciones a realizar en segundo plano.

Ejemplo:

class MyBackgroundTask extends Thread {

@Override

public void run() {

// Realizar operaciones en segundo plano

// Evitar interacciones directas con la interfaz de usuario

}

}

* Actualizaciones de Interfaz en el UI Thread:
  + La interfaz de usuario en Android está gestionada por el UI Thread. Cualquier modificación en las vistas de la interfaz debe realizarse en este hilo para evitar conflictos y mantener la fluidez de la aplicación.
  + Actualizar la interfaz directamente desde un *worker-thread* puede resultar en errores y bloqueos debido a la naturaleza no sincronizada de las operaciones de interfaz.
* Utilización de Handler o runOnUiThread() para Modificar la Interfaz:
  + Se utilizan mecanismos como Handler o runOnUiThread() para comunicarse con el UI Thread y realizar modificaciones en las vistas de la interfaz de manera segura.
  + Handler: Proporciona una forma de programar tareas para que se ejecuten en el UI Thread desde otros threads. Se asocia con el Looper y MessageQueue del hilo en el que se crea.
  + runOnUiThread(): Un método de la clase Activity que permite ejecutar código en el UI Thread desde un *worker-thread*.
* Ejemplo con Handler:
* java
* Copy code

Handler uiHandler = new Handler(Looper.getMainLooper());

// En el worker-thread

uiHandler.post(new Runnable() {

@Override

public void run() {

// Modificar la interfaz de usuario de manera segura

}

});

* Ejemplo con runOnUiThread():
* java
* Copy code

runOnUiThread(new Runnable() {

@Override

public void run() {

// Modificar la interfaz de usuario de manera segura

}

});

Estos mecanismos aseguran una comunicación segura entre threads y permiten que las operaciones intensivas se realicen en segundo plano sin afectar la responsividad del UI Thread.

**Ejemplo de Actualización de Interfaz en Android:**

private void yourMethodName() {

new Thread(new Runnable() {

@Override

public void run() {

try {

yourActivity.runOnUiThread(new Runnable() {

@Override

public void run() {

txtview.setText("some value");

edittext.setText("some new value");

}

});

} catch (Exception e) {

// Manejar el error aquí

}

}

}).start();

***Clase Handler y Runnables: Simplificando Conceptos***

## Clase Handler

La biblioteca Android ofrece diversas herramientas para facilitar la gestión de hilos en las aplicaciones. Una de ellas son los Handler.

La clase Handler está diseñada para dar soporte a la gestión del trabajo entre dos threads cualesquiera, no siendo el UI Thread necesariamente uno de ellos, y cada Handler está asociado con un thread específico.

## Transferir tareas a otro proceso

Un thread puede transferir trabajo para otro thread mediante:

**-El envío de un Message:** Un Message es una clase que puede contener datos tales como un código de mensaje, un objeto arbitrario o valores enteros. Se usan por el thread emisor para indicarle al receptor qué operación realizar, dejando al Handler la implementación de ésta.

**-El envío de un Runnable:** Los Runnables los usamos cuando sabemos exactamente cuáles son pasos de ejecución a realizar, pero queremos que sean ejecutados en el thread receptor.

## Runnables

Los Handlers gestionan los Runnables haciendo uso de dos componentes asociados al thread, un *MessageQueue* y un *Looper* .

### Looper

Looper actúa como un intermediario que toma mensajes de la cola y los envía al componente correcto para su procesamiento, asegurando que las operaciones se realicen de manera ordenada y secuencial. Esto es fundamental para evitar condiciones de carrera y garantizar un comportamiento predecible en la ejecución de hilos.

Un Looper puede tener vinculados uno o varios Handlers. Estos Handlers, pueden ser del mismo productor o de distintos productores.

### MessageQueue

Es una lista enlazada ilimitada de mensajes para procesar en el hilo del consumidor. Cada Looper, y Thread, tiene como máximo un MessageQueue.

Cuando se crea un thread, ya sea el thread principal creado por Android para nuestra aplicación o uno que creamos con new Thread(), se crean también su Looper y su MessageQueue.

Cuando se crea un Handler con new Handler(), éste se vincula al MessageQueue y al Looper de la aplicación o activity en el que se crea.

Por tanto, un thread tiene finalmente asociados un Handler, un Looper y una MessageQueue que a su vez tiene asociados Mensajes, siendo el Handler el origen y destino de los mensajes de la cola.

El Looper está continuamente esperando la llegada de trabajos al MessageQueue. Estos trabajos pueden ser Messages o Runnables. Cuando los trabajos llegan, el Looper reacciona en función del tipo de trabajo de que se trate.

En el contexto del manejo de hilos en Android, estas acciones son parte del flujo de trabajo entre productores (o emisores de tareas) y consumidores (o receptores de tareas) de hilos. Aquí está cómo estas acciones son realizadas y su objetivo:

* Insert (Inserción):
  + Quién: El productor o emisor de la tarea, que podría ser cualquier thread, utiliza un Handler conectado al thread consumidor.
  + Objetivo: Enviar una tarea (Runnable) al thread consumidor para su ejecución.
* Retrieve (Recuperación):
  + Quién: El Looper, que se ejecuta en el thread consumidor, recupera mensajes de la cola secuencialmente.
  + Objetivo: Preparar mensajes, incluidos los Runnables, para su ejecución.
* Dispatch (Despacho):
  + Quién: El Handler asociado al Looper.
  + Objetivo: Ejecutar la tarea asociada al mensaje recuperado. En el caso de Runnables, esto implica llamar al método run() del Runnable para llevar a cabo la tarea específica en el hilo consumidor.

El objetivo general es permitir la comunicación y transferencia de tareas entre diferentes hilos, facilitando la ejecución de operaciones específicas en un contexto controlado y evitando problemas como bloqueos en la interfaz de usuario.

En cuanto a la interacción con el Thread principal, si queremos interactuar con el UI Thread, enviándole Runnables) por ejemplo, tenemos que instanciar un Handler en el contexto de la aplicación o Activity.

Cuando hacemos new Handler() en el onCreate() de una Activity, este nuevo Handler se está vinculando con el Looper y el MessageQueue del UI Thread, porque es en su contexto donde se está creando el handler. Por tanto cualquier runnable o message que enviemos a través de los métodos .post() del handler, serán ejecutados en el UI Thread cuando proceda.

Si un thread quiere ejecutar un código, entonces crea u obtiene un Runnable, donde el código a ejecutar está su método run() para que su handler, el que está asociado al thread A, lo ejecute, es decir, que sea el propio thread el que lo ejecute, no el UI Thread u otro thread. Para ello debe entregarlo (postearlo) a su Handler asociado, que lo insertará en su cola de mensajes, es decir, al Handler que se ha creado en su clase (o que se ha vinculado a su Looper).

### **Métodos utilizados para enviar Runnables a un Handler:**

Añade el Runnable al MessageQueue

boolean post(Runnable r)

Añade el Runnable al MessageQueue para ser ejecutado en un instante concreto.

boolean postAtTime(Runnable r, long uptimeMilis)

Añade el Runnable al MessageQueue para ser ejecutado pasado un tiempo definido.

boolean postDelayed(Runnable r, long uptimeMilis)

# ***Ejercicio de Ejemplo***

Vamos a hacer un ejemplo donde vamos a ver la comunicación entre threads mediante el uso de Handlers en Android.La interfaz tendrá 2 botones y un textview que se cambia según el evento. Cargaremos una imagen en segundo plano mientras se muestra el progreso en un TextView. Esto lo hacemos con el objetivo de mostrar el uso de hilos en Android para realizar tareas en segundo plano y la necesidad de utilizar un Handler para actualizar la interfaz de usuario desde esos hilos secundarios.

1. **Layout (main.xml):**
   * Se espera que este archivo contenga la estructura de la interfaz de usuario. El código se refiere a elementos de la interfaz de usuario utilizando sus identificadores (IDs), como imageView, textView, etc.
2. **HandlerRunnableActivity (Clase principal):**
   * La actividad principal de la aplicación que extiende Activity.
   * Incluye variables miembro para ImageView, TextView, Bitmap, y un Handler.
   * El Handler se utiliza para realizar operaciones en el hilo de interfaz de usuario desde otros hilos.
3. **LoadIconTask (Clase interna Runnable):**
   * Implementa la lógica de carga de la imagen y la simulación de un proceso largo.
   * Dentro del método run:
     + Muestra "Running" en el TextView al inicio del hilo.
     + Itera 10 veces para simular un proceso largo, actualizando el TextView con el progreso.
     + Carga la imagen desde recursos y la muestra en un ImageView.
     + Establece el TextView de nuevo a "Not running yet" al finalizar el hilo.
4. **Método onClick para los botones:**
   * Se encarga de manejar los clics en los botones.
   * Para el botón "Clear" (btnClear), limpia la imagen en el ImageView.
   * Para el botón "LoadIcon" (button), inicia un nuevo hilo (Thread) ejecutando LoadIconTask.
5. **Método sleep:**
   * Introduce un retraso en la ejecución del hilo para simular un proceso largo.

package course.examples.ThreadingHandlerRunnable;

import android.app.Activity;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.os.Bundle;

import android.os.Handler;

import android.os.Looper;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.ImageView;

import android.widget.TextView;

import android.widget.Toast;

public class HandlerRunnableActivity extends Activity {

private ImageView mImageView;

private TextView mTextView; // Cambiamos a TextView

private Bitmap mBitmap;

private int mDelay = 500;

private Handler handler = new Handler();

private int progressCounter = 0;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

mImageView = findViewById(R.id.*imageView*);

mTextView = findViewById(R.id.*textView*); // Cambiamos a TextView

// Botón Clear

final Button btnClear = findViewById(R.id.*btnClear*);

// Definimos el manejador del OnClick del boton clear

btnClear.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

// Quitamos la imagen

mImageView.setImageBitmap(null);

}

});

final Button button = findViewById(R.id.*loadButton*);

button.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

Thread newThread = new Thread(new LoadIconTask(R.drawable.*painter*));

newThread.start();

}

});

final Button otherButton = findViewById(R.id.*otherButton*);

otherButton.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

Toast.*makeText*(HandlerRunnableActivity.this, "I'm Working",

Toast.*LENGTH\_SHORT*).show();

}

});

}

private class LoadIconTask implements Runnable {

int resId;

LoadIconTask(int resId) {

this.resId = resId;

}

public void run() {

handler.post(new Runnable() {

@Override

public void run() {

mTextView.setText("Running");

}

});

mBitmap = BitmapFactory.*decodeResource*(getResources(), resId);

// Simulating long-running operation

for (int i = 1; i < 11; i++) {

sleep();

final int step = i;

handler.post(new Runnable() {

@Override

public void run() {

mTextView.setText("Running " + step + " time");

}

});

}

handler.post(new Runnable() {

@Override

public void run() {

mImageView.setImageBitmap(mBitmap);

mTextView.setText("Not running yet");

}

});

}

}

private void sleep() {

try {

Thread.*sleep*(mDelay);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/textView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<ImageView

android:id="@+id/imageView"

android:layout\_width="400dp"

android:layout\_height="340dp"

android:scaleType="centerInside"></ImageView>

<ProgressBar

android:id="@+id/progressBar"

style="@android:style/Widget.ProgressBar.Horizontal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:maxHeight="5dip"

android:minHeight="5dip"

android:visibility="invisible"></ProgressBar>

<Button

android:id="@+id/loadButton"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Load Icon"

android:textSize="24sp"></Button>

<Button

android:id="@+id/otherButton"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Other Button"

android:textSize="24sp"></Button>

<Button

android:id="@+id/btnClear"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Clear" />

</LinearLayout>

DIAGRAMA DE INTERACCIONES ENTRE EL HILO PRINCIPAL Y EL SECUNDARIO:Puedes crear un diagrama sencillo para visualizar el flujo de ejecución entre el hilo principal y el hilo de fondo en la Forma 1. Puedes seguir estos pasos para representar el flujo:

* Inicio:
  + Coloca un rectángulo que represente el inicio del programa. Etiqueta este rectángulo como "Inicio".
* Botón "Load Icon" Presionado:
  + Agrega una flecha desde el "Inicio" hacia un rectángulo que represente el evento de presionar el botón "Load Icon".
  + Dentro de este rectángulo, escribe "onClick()" y "newThread.start()".
* Creación del Thread (hilo de fondo):
  + Añade una flecha que representa la creación del nuevo hilo (Thread) desde el rectángulo anterior.
  + Etiqueta este paso con "New Thread" y agrega "LoadIconTask(R.drawable.painter)".
* Inicio del LoadIconTask:
  + Agrega una flecha que representa el inicio del LoadIconTask dentro del nuevo hilo.
  + Dentro de este paso, indica la creación de un nuevo Handler y la ejecución de un Runnable para actualizar el TextView.
* Decodificación de Recurso y Tarea Larga:
  + Representa la decodificación del recurso (BitmapFactory.decodeResource()) y la simulación de la tarea larga (bucle de 1 a 10) dentro del LoadIconTask.
* Actualización de la Interfaz de Usuario:
  + Agrega flechas que representen las actualizaciones de la interfaz de usuario utilizando Handler.post() y Runnable en diferentes etapas de la tarea.
* Fin del LoadIconTask:
  + Añade una flecha que represente el final del LoadIconTask y cómo se muestra la imagen en el ImageView al finalizar la tarea.
* Fin:
  + Coloca un rectángulo que represente el final del programa. Etiqueta este rectángulo como "Fin".

Este diagrama proporcionará una visualización simple y clara del flujo de ejecución en la Forma 1. Puedes utilizar herramientas como papel y lápiz, pizarra, o software de dibujo y diagramación para crear este diagrama. Recuerda que el objetivo es comunicar de manera efectiva el flujo de ejecución del programa a través de los hilos.

VER ERRORES QUE PUEDE DAR SI NO SE GESTIONAN HILOS

MOSTRAR DIFERENCIAS D EHACERLO CON LOS 2 CODIGOS

PONER CODIGOS

RESUMIR-->NO TANTA EXPLICACION EN LOS POWERPOINT

EN LOS METODOS->PONER EJEMPLO DE CODIGO

NAVIGATION DRAWER EN LUGAR DE BOTONES