LifeCycles and Intents (Parte II)

Integración de datos entre Actividades y Fragmentos

Competencias:

- Enviar datos desde una actividad hacia un fragmento
- Recibir datos de una actividad en un fragmento
- Enviar datos desde un fragmento hacia una actividad
- Recibir datos de un fragmento en una actividad
- Enviar datos entre fragmentos

Introducción

En este capítulo explicaremos los métodos tradicionales de envío de datos entre actividades y fragmentos y dedicaremos esfuerzos a la realización de cuatro ejercicios que nos ayudarán a entender de mejor manera la integración de datos y también a desarrollar nuestro proyecto principal del curso “PruebaDinámica”.

Iniciando un Fragmento con datos desde la actividad

Basándonos en nuestros ejercicios anteriores y en nuestro proyecto “Prueba Dinámica” pasaremos a explicar el siguiente código:

```
PreguntaFragment preguntaFragment = PreguntaFragment
    .newInstance(
        pregunta.getQuestion(),
        pregunta.getCategory(),
        pregunta.getCorrect_answer(),
        pregunta.getIncorrect_answers()
    );
```

La anterior es una de las formas más elegantes de pasar datos desde una actividad hacia un fragmento, instanciando nuestra clase fragmento a través de un método que emula un método constructor, el newInstance(), que recibe parámetros, en este caso la pregunta, la categoría, la respuesta correcta y las respuestas incorrectas de nuestra api de datos.

La declaración de este método en nuestra clase fragmento es la siguiente:

```
public static PreguntaFragment newInstance(String pregunta,
                                           String categoria,
                                           String respuestaCorrecta,
                                           ArrayList<String> respuestasIncorrectas) {
    PreguntaFragment fragment = new PreguntaFragment();
    Bundle arguments = new Bundle();
    arguments.putString("PREGUNTA", pregunta);
    arguments.putString("CATEGORIA", categoria);
    arguments.putString("RESPUESTA_CORRECTA", respuestaCorrecta);
    arguments.putStringArrayList("RESPUESTAS_INCORRECTAS", respuestasIncorrectas);
    fragment.setArguments(arguments);

    return fragment;
}
```

Del código anterior, se destaca que es el método newInstance de la clase quién hace público una instancia del fragmento y crea los argumentos en base a los parámetros recibidos, para que en una fase posterior del ciclo de vida como lo es desde el onCreateView() , sean rescatados los datos para su uso asociado a las vistas.

Este patrón llamado “newInstance” es aplicado en muchas aplicaciones android, principalmente por ser una de las formas más limpias de iniciar un fragmento desde cualquier actividad y también por mantener los parámetros que recibe el fragmento cuando la actividad se detiene y se vuelve a activar. En el caso de un constructor nativo java esto no ocurre ya que android llama por defecto al constructor sin argumentos de una clase cada vez que esta es iniciada.

Bundle

Es una clase que permite asociar datos o mapearlos para su posterior uso. Básicamente permite agregar objetos declarando una llave de tipo String y un valor. Generalmente se les conoce como argumentos y estos se identifican con dos tipos de métodos:

- **setArguments():** método utilizado para guardar los datos declarados como argumentos en la instancia de clase Bundle del fragmento.
- **getArguments():** método utilizado para capturar los datos desde el fragmento que han sido enviados por una actividad, a través de una llave de tipo String.

Enviar datos a un fragmento específico cuando ya están creados

Una actividad que contiene múltiples fragmentos ya creados, puede entregar mensajes a uno de estos mediante la captura de la instancia Fragment con findFragmentById() y luego invocar directamente a los métodos públicos del fragmento.

```
getSupportFragmentManager().findFragmentById(R.id.fragment_pregunta);
```

Enviar datos desde un fragmento hacia la actividad que lo contiene

Cuando trabajamos con fragmentos es común que necesitemos capturar datos desde nuestra actividad de un fragmento que será detenido o destruido para dar paso a otro fragmento o posiblemente a otra actividad.

Una de las formas más comunes de enviar datos a una actividad desde un fragmento es la creación de una interfaz auxiliar que sea implementada en ambas clases tanto la activity como la del fragmento de la siguiente forma:

- Se crea una clase interfaz que contenga los métodos y atributos que requerimos.
- Se implementa la interfaz en la actividad y los métodos que requerimos.
- En el fragmento utilizando el método onAttach() y onDetach() del ciclo de vida, declaramos el listener asociado a dicha interfaz.

  Asignamos los valores a través del método de la interfaz desde nuestro fragmento.

De esta manera, podremos recibir en nuestra actividad la información que se registra en nuestro fragmento para así poder enviarla a otro fragmento u otra actividad. En el próximo ejercicio realizaremos esta tarea.

Ejercicio 6:

Enviar datos desde nuestro fragmento a nuestra actividad a través de una pequeña interfaz, implementada en la actividad e instanciada en el fragmento. La actividad debe mostrar un mensaje Toast desde el onactivityResult con los datos recibidos desde el fragmento.

1. En nuestra carpeta fragment, crearemos una pequeña interfaz llamada “OnFragmentPreguntaListener “, la cual se encargará de comunicar los datos de selección de respuesta entre nuestro fragmento y actividad:

```
package cl.desafiolatam.pruebadinamica.fragments;

public interface OnFragmentPreguntaListener {
    void onRadioButtonSelection(int selection, String respuesta);
}
```

2. En nuestra clase “MainActivity” es necesario que implementemos esta nueva interfaz de la siguiente forma:

```
public class MainActivity extends AppCompatActivity implements OnFragmentPreguntaListener {
```

Lo que a su vez, nos conlleva a implementar los métodos de esta interfaz en nuestra actividad, agregando el siguiente método a nuestra clase “MainActivity”:

```
@Override
public void onRadioButtonSelection(int selection, String respuesta) {
   Toast.makeText(getApplicationContext(),
         "La respuesta seleccionada en el fragmento fue: la: "+selection+" , correspondiente a "+respuesta,
         Toast.LENGTH_SHORT).show();
   Log.d(TAG, "La respuesta seleccionada en el fragmento fue la: "+selection+" , correspondiente a
"+respuesta);
}
```

4. Declaramos una variable de clase en nuestro fragmento “PreguntaFragmento” que nos comunicará con nuestra actividad de la siguiente forma:

```
//LISTENER DE INTERFAZ QUE COMUNICA NUESTRO FRAGMENTO CON LA ACTIVIDAD CONTENEDORA
private OnFragmentPreguntaListener mListener;
```

5. Recordando el capítulo “Ciclo de vida de un fragmento”, vamos a sobreescribir el método onAttach() para que al momento de agregarse nuestro fragmento al contexto, este agregue también una declaración de la interfaz “OnFragmentPreguntaListener” de la siguiente manera:

```
@Override
public void onAttach(Context context) {
  super.onAttach(context);
  if (context instanceof OnFragmentPreguntaListener) {
     mListener = (OnFragmentPreguntaListener) context;
  } else {
     throw new ClassCastException(context.toString()
           + "Aún no implementas la interfaz en la actividad");
  }
}
```

6. A continuación editaremos el método “onCheckedChanged” que asigna valores a los radio buttons según la selección aplicada de la siguiente manera:

```
@Override
public void onCheckedChanged(RadioGroup group, int checkedId) {
  if (respuestaUno.isChecked()) {
     setRadioButtonValues(1, respuestaUno.getText().toString());
  } else if (respuestaDos.isChecked()) {
     setRadioButtonValues(2, respuestaDos.getText().toString());
  } else if (respuestaTres.isChecked()) {
     setRadioButtonValues(3, respuestaTres.getText().toString());
  } else if (respuestaCuatro.isChecked()) {
     setRadioButtonValues(4, respuestaCuatro.getText().toString());
  }
}
```

7. A continuación declaramos en nuestra clase el método setRadioButtonValues(), el cual se encargará de enviar hacia el listener de la interfaz el número de respuesta seleccionado, así como también el texto de su respuesta, esto para que nuestra actividad pueda recibir estos datos y usarlos según sea requerido:

```
private void setRadioButtonValues(int radioButtonValue, String respuesta) {
  //ENVIAMOS NUESTRO VALOR SELECCIONADO AL LISTENER DE LA INTERFAZ
  mListener.onRadioButtonSelection(radioButtonValue, respuesta);
  this.radioButtonValue = radioButtonValue;
}
```

8. Por último agregamos un método más del ciclo de vida del fragmento para que al destruirse el fragmento sea desconectada la interfaz declarada de la siguiente manera:

```
@Override
public void onDetach() {
  mListener = null;
  super.onDetach();
}
```

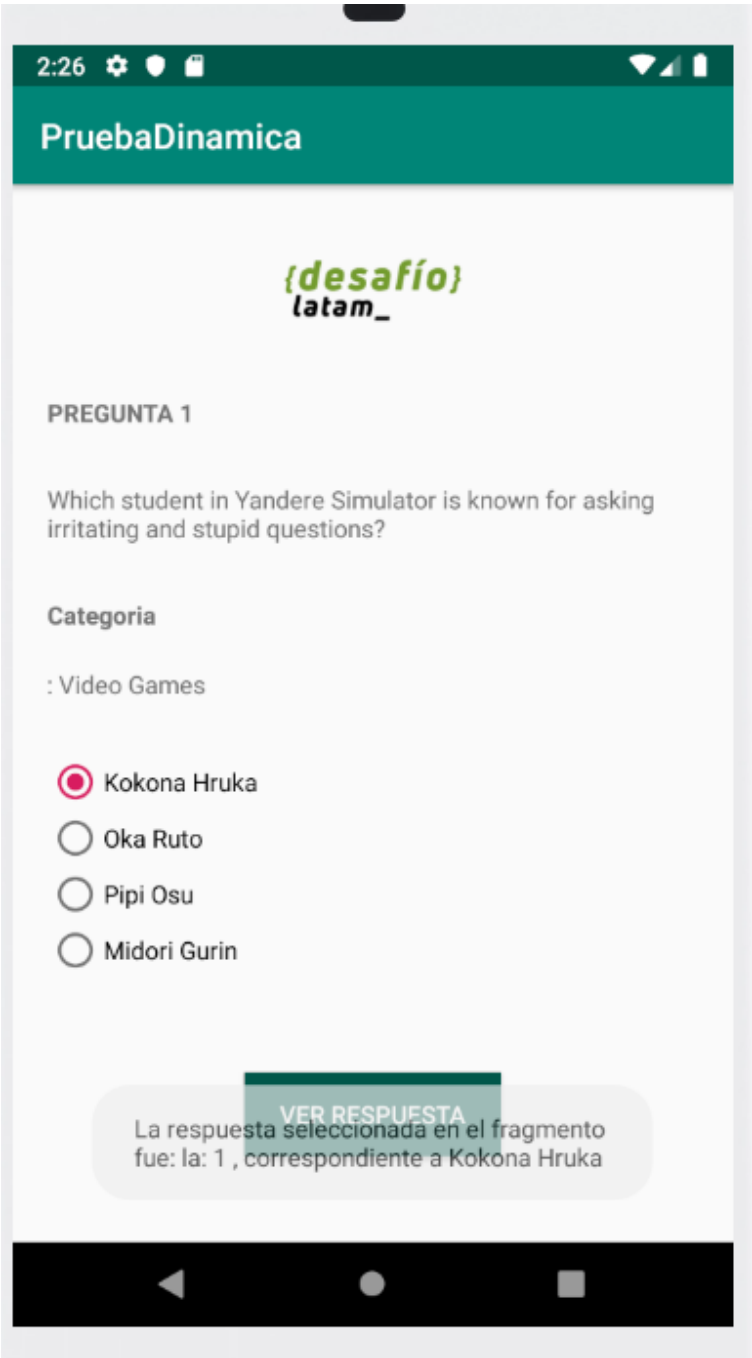

Imagen 1: Resultado Ejercicio 6

Ejercicio 7:

Construir múltiples fragmentos a partir del arreglo results de nuestra api, que al deslizar la pantalla hacia la derecha o izquierda se refresque la pregunta hasta un total de diez.

1. Nos ubicamos en nuestra carpeta cl.desafiolatam.pruebadinamica y creamos un nuevo package de nombre “adapters”.
2. Ubicados en adapters, vamos a crear una nueva clase java tipo adapter de nombre “FragmentosPageAdapter”, cuya función será manejar una lista de fragmentos, capturar su posición y retornar el fragmento solicitado a la pantalla.
3. La clase debe extender para este ejemplo de la clase FragmentPagerAdapter, debe declarar una variable de clase tipo List, declarar un constructor que reciba una lista de fragmentos y pase ese objeto a su variable de clase, además de implementar los métodos getCount y getItem propios de la clase extendida. La clase sería la siguiente:

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import java.util.List;

public class FragmentosPageAdapter extends FragmentPagerAdapter {

    private List<Fragment> fragmentosLista;

    public FragmentosPageAdapter(FragmentManager fm, List<Fragment> fragmentos){
        super(fm);
        this.fragmentosLista = fragmentos;
    }

    @Override
    public int getCount() {
        return this.fragmentosLista.size();
    }

    @Override
    public Fragment getItem(int position) {
        return this.fragmentosLista.get(position);
    }
}
```

4. En nuestro archivo layout “activity_main.xml” reemplazamos el FrameLayout por el siguiente viewgroup:

```
<android.support.v4.view.ViewPager
  android:id="@+id/viewpager"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  />
```

5. En nuestro fragmento “PreguntaFragmento” realizaremos unas pequeñas modificaciones de la siguiente manera:

Al método constructor le agregaremos un nuevo parámetro llamado numeroPregunta, con el cual sabremos a qué número de pregunta pertenece el fragmento. En el siguiente código se muestran destacados en amarillo los cambios a realizar.

```
public static PreguntaFragment newInstance(int numeroPregunta,
                                           String pregunta,
                                           String categoria,
                                           String respuestaCorrecta,
                                           ArrayList<String> respuestasIncorrectas) {
    PreguntaFragment fragment = new PreguntaFragment();
    Bundle arguments = new Bundle();
    arguments.putString("NUMERO_PREGUNTA", "PREGUNTA "+numeroPregunta);
    arguments.putString("PREGUNTA", pregunta);
    arguments.putString("CATEGORIA", categoria);
    arguments.putString("RESPUESTA_CORRECTA", respuestaCorrecta);
    arguments.putStringArrayList("RESPUESTAS_INCORRECTAS", respuestasIncorrectas);
    fragment.setArguments(arguments);

    return fragment;
}
```

6. En nuestro método OnCreateView del mismo fragmento capturamos este nuevo argumento que incluimos en el constructor, y además le asignamos el valor a la vista correspondiente.

```
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable
Bundle savedInstanceState){

    View view = inflater.inflate(R.layout.fragment_pregunta, container, false);

    final String preguntaTitulo = Objects.requireNonNull(getArguments()).getString("NUMERO_PREGUNTA");
    final String pregunta = Objects.requireNonNull(getArguments()).getString("PREGUNTA");
```

7. En el mismo método onCreateView y después del llamado al método initializeViews, asignamos el valor rescatado de los argumentos a la vista correspondiente.

```
preguntaLabel.setText(preguntaTitulo);
```

8. En el método initializeViews, agregamos la siguiente instrucción:

```
preguntaLabel = view.findViewById(R.id.tituloPregunta);
```

9. En nuestra clase de actividad “MainActivity”, agregamos dos nuevas variables de clase, una declarada para el uso del adaptador FragmentosPageAdapter, y la otra para el uso de nuestra nuevo groupview el ViewPager.

```
FragmentosPageAdapter fragmentsPageAdapter;
ViewPager viewPager;
```

10. En la clase MainActivity, crearemos un nuevo método cuya labor será construir una lista de instancias de fragmentos para ser enviada posteriormente a nuestra instancia del adaptador, el cual se encargará de asignar una ubicación a cada uno de estos fragmentos.

Para lograr esto, creamos el método getFragmentosDinamicos(PreguntasLista preguntas) al final de nuestra clase de actividad, replicando el código “*PreguntaFragment.newInstance”* de nuestro método onCreateView, adaptándolo a este nuevo comportamiento dentro de una iteración de resultados, de la siguiente manera:

```
private List<Fragment> getFragmentosDinamicos(PreguntasLista preguntas){

  List<Fragment> fragmentosLista = new ArrayList<>();
  int contadorPregunta = 1;

  for(Pregunta pregunta: preguntas.getResults()) {
    fragmentosLista.add(
        PreguntaFragment
            .newInstance(
                contadorPregunta,
                pregunta.getQuestion(),
                pregunta.getCategory(),
                pregunta.getCorrect_answer(),
                pregunta.getIncorrect_answers()
            )
    );
    contadorPregunta++;
  }
  return fragmentosLista;
}
```

11. El punto 5 del ejercicio 5 de nuestro capítulo “Ciclo de Vida de un Fragmento” se nos indicaba cómo iniciar la instancia del fragmento a partir de la respuesta del api de datos. En esta ocasión vamos a editar en ese mismo método onResponse, para que en vez de abrir un solo fragmento a partir del primer resultado de nuestro arreglo results del api, sean creados tantos objetos como resultados obtengamos, entendiendo que si al consultar nuestra api de datos el resultado son 10 objetos, 10 fragmentos serán creados a partir de ahora con sus datos propios. Para continuar nuestro método callback onResponse dentro de nuestra clase MainActivity debe quedar de la siguiente forma:

```
//Async
call.enqueue(new Callback<PreguntasLista>() {
  @Override
  public void onResponse(Call<PreguntasLista> call, Response<PreguntasLista> response) {
        Log.d(TAG, response.toString());
        PreguntasLista preguntas = response.body();

    if (preguntas != null) {

        List<Fragment> paginasFragmentos = getFragmentosDinamicos(preguntas);
        fragmentsPageAdapter = new FragmentosPageAdapter(getSupportFragmentManager(),
paginasFragmentos);
        viewPager = findViewById(R.id.viewpager);
        viewPager.setAdapter(fragmentsPageAdapter);

    }
  }
  @Override
  public void onFailure(Call<PreguntasLista> call, Throwable t) {
    Toast.makeText(getApplicationContext(), "Error: no pudimos recuperar los datos de opentdb",
Toast.LENGTH_SHORT).show();
  }
});
```

De lo anterior, destacamos en amarillo los cambios realizados sobre el método, reafirmando que la intención del cambio es capturar una lista de fragmentos, enviar esta lista a una clase adaptador que asigna a cada uno de ellos espacio e identificación en nuestra aplicación, para finalmente indicar a nuestro viewgroup ViewPager que el adaptador es el anterior.

12. Para finalizar tendremos una aplicación con 10 fragmentos cada uno representativo de una pregunta con sus respuestas propias con datos externos provistos por opendtb.com.

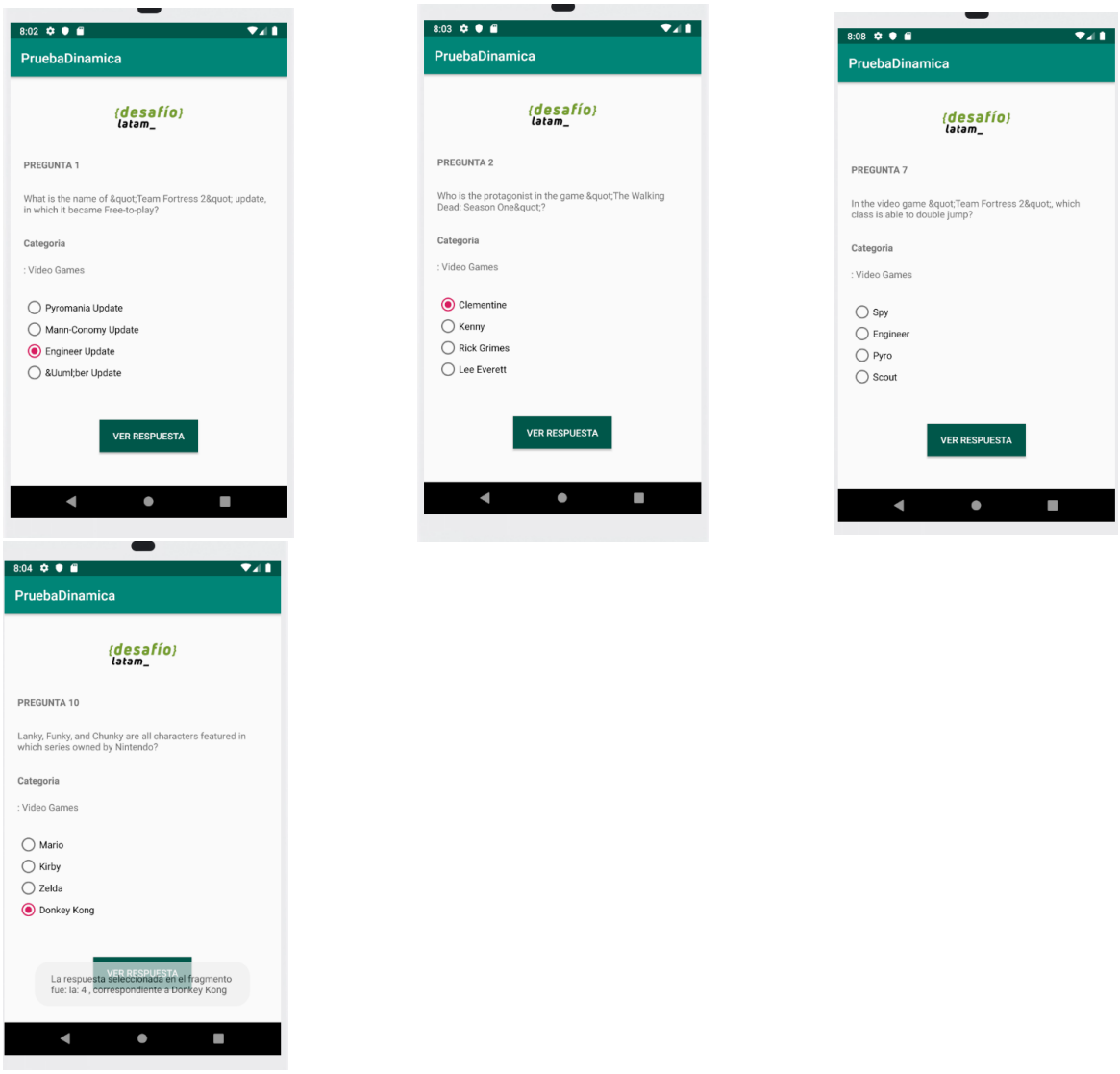

Imagen 2: Resultado Ejercicio 7

Envío de datos entre fragmentos

El compartir datos entre un fragmento A y un fragmento B, es un ejercicio similar al practicado en el ejercicio 6 del capítulo en curso referido al envío de datos desde un fragmento hacia una actividad.

Para enviar datos de un fragmento a otro, se pueden implementar los siguientes pasos:

Crear una interfaz que servirá de enlace entre el fragmento a y el fragmento b.

```
package cl.desafiolatam.pruebadinamica.fragments;

public interface OnFragmentPreguntaListener {
    void onRadioButtonSelection(int selection, String respuesta);
}
```

Implementar la interfaz en el fragmento receptor del mensaje.

Declarar una variable de clase tipo listener de la interfaz en el fragmento emisor.

En el método onAttach() del fragmento emisor, instanciar el listener de la interfaz.

```
@Override
public void onAttach(Context context) {
  super.onAttach(context);
  if (context instanceof OnFragmentPreguntaListener) {
      mListener = (OnFragmentPreguntaListener) context;
  } else {
      throw new ClassCastException(context.toString()
              + "Aún no implementas la interfaz en la actividad");
  }
}
```

Intents, implícitos y explícitos

Competencias:

- Desarrollar intents
- Conocer distintos tipos de intents
- Diferenciar intents implícitos de intents explícitos

Introducción

Los intents son los responsables de comunicar nuestros componentes entre sí, son fundamentales en el funcionamiento de una aplicación android. Un intent nos permite iniciar una actividad, invocar un servicio, enviar o recibir mensajes, abrir la cámara de nuestro teléfono, abrir una página web en el navegador, entre muchas otras acciones tanto dentro de nuestra propia aplicación como en comunicación con otras aplicaciones externas.

Intents

Un Intent es un objeto que el sistema operativo android utiliza para iniciar un determinado componente que puede estar dentro o fuera de nuestra aplicación. Un Intent también se le conoce como un objeto que realiza una acción o solicita una acción de otro componente de la aplicación.

Es recurrente el uso de los intents para tres casos en particular:

- **Iniciando una actividad:** una Actividad representa una única pantalla en una aplicación. Puedes iniciar una nueva instancia de una Actividad pasando un Intent a través del método startActivity(). El Intent describe la actividad que se debe iniciar y contiene datos necesarios para ser capturados por el componente receptor.
- **Un Servicio**: es un componente que realiza operaciones en segundo plano sin una interfaz de usuario (pantalla). Un servicio puede ser utilizado para realizar una operación única, como por ejemplo, descargar un archivo, pasando un Intent al método startService(). El Intent describe el servicio que se debe iniciar y contiene los datos necesarios para el componente receptor.
- **Entregar un mensaje:** un mensaje es un aviso que cualquier aplicación puede recibir (share intent). El sistema entrega varios mensajes de eventos del sistema, como cuando el sistema arranca o el dispositivo comienza a cargarse. Para enviar un mensaje a otras aplicaciones podemos usar un Intent a través del método sendBroadcast().

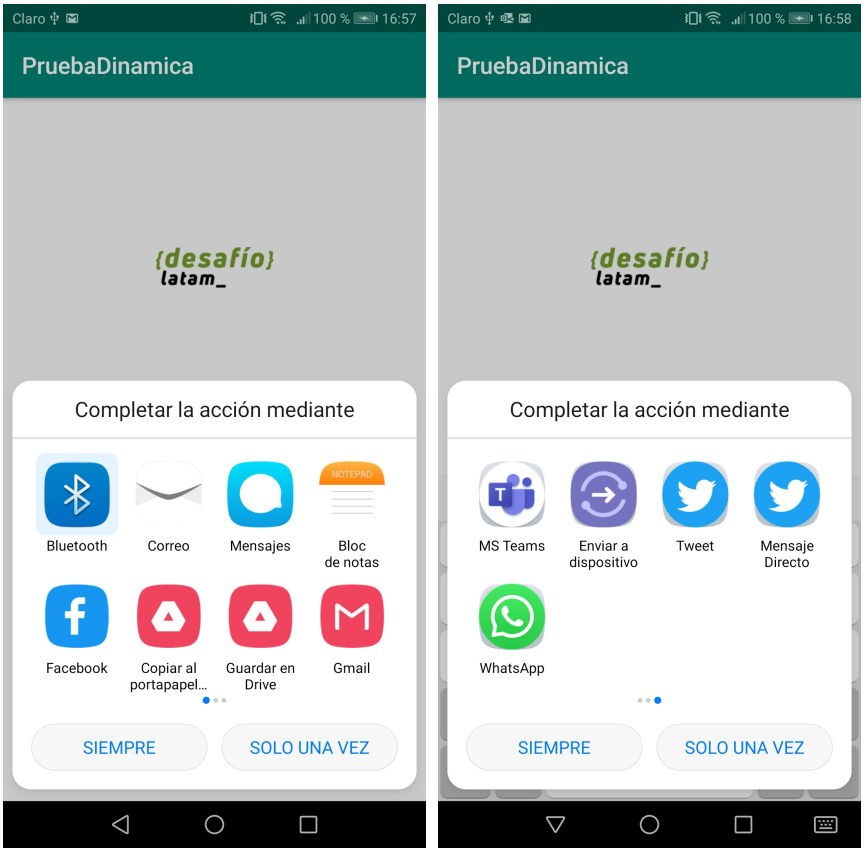

Imagen 3: Share intent (compartir)

Intents Implícitos y Explícitos

Los intents se pueden clasificar en intents implícitos e intents explícitos, principalmente diferenciados por un parámetro opcional que asigna el nombre del componente a llamar, es decir, los intents implícitos no indican el nombre de su componente destino, como ocurre en el caso de abrir una web en el navegador; mientras que los intents explícitos suelen ser aquellos en los que nos comunicamos con un componente dentro de nuestra propia aplicación como sería el caso de iniciar una segunda actividad, declarando el nombre del componente “SegundaActividad.java”.

Implícitos

Son aquellos en los que no se especifica un nombre de componente, en su lugar, se especifica la funcionalidad requerida a través de una acción que según su clasificación, indica automáticamente al sistema operativo android qué software debe ejecutar la petición, como es el caso de abrir la pantalla de marcar un teléfono, abrir la lista de contactos, un navegador, una cámara, etc… En un intent implícito también se especifican datos asociados a estas acciones, como por ejemplo, la dirección web de la página que abrirá el navegador o el número de teléfono para hacer una llamada.

Es muy común asociar intents implícitos al consumo de aplicaciones propias del sistema operativo. A continuación algunos intents implícitos más comunes nos permiten interactuar con este tipo de componentes:

- Alarmas
- Mensajería de texto
- Cámara
- Aplicación de Música
  Aplicación de Videos
- Calendarios
- Notas
- Correo electrónico

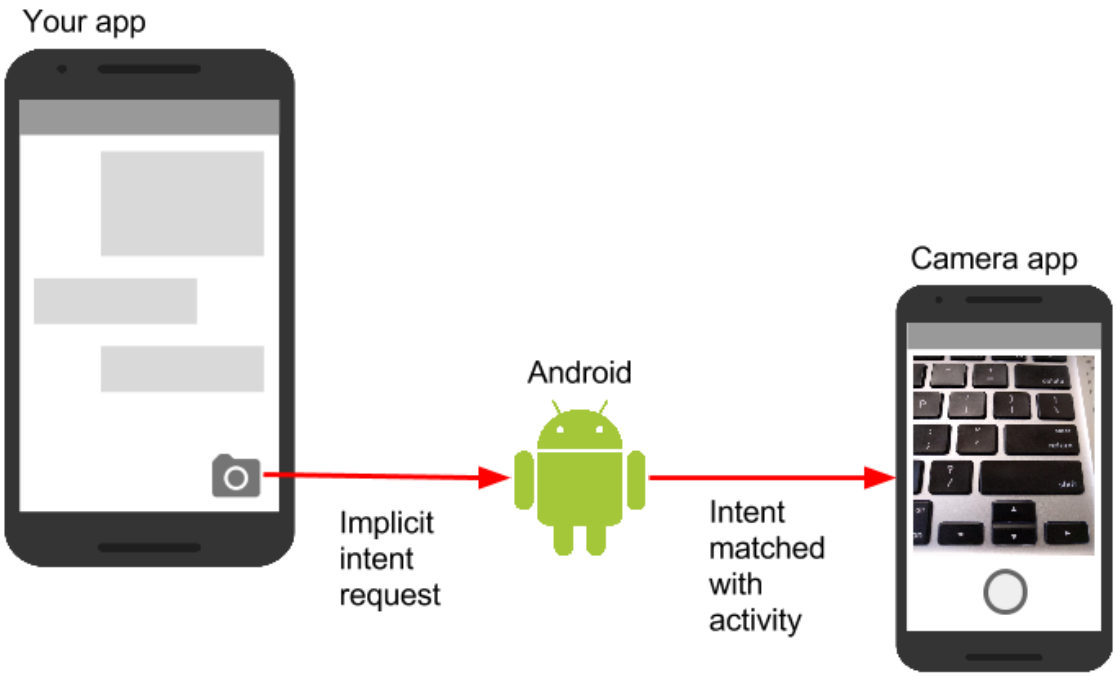

Imagen 4: Intent implícito

Intent implícito de cámara

Si tu aplicación requiere del uso de la cámara, debemos indicar esto en nuestro archivo manifest con el tag de la siguiente manera:

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

Debemos crear una variable de clase con el código de request que queramos indicar y un método que realice el intent hacia el componente de la cámara de la siguiente manera:

```
static final int REQUEST_IMAGE_CAPTURE = 1;

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}
```

Para asociar la foto tomada a un imageview dentro de nuestra aplicación, debemos hacer la asignación desde el método onActivityResult, una vez se haya ejecutado el intent de cámara, de la siguiente forma:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        imageView.setImageBitmap(imageBitmap);
    }
}
```

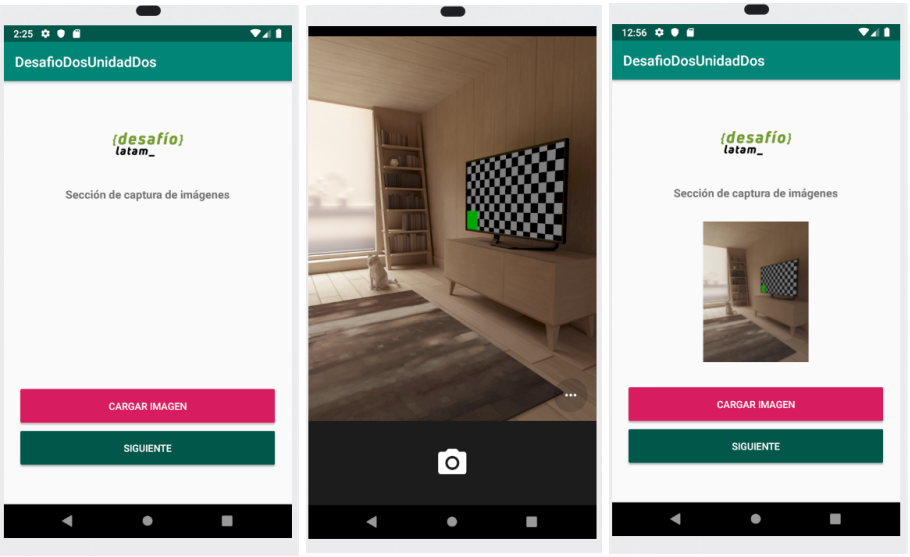

Imagen 5: Intent cámara

Explícitos

Son aquellos componentes que residen generalmente dentro de nuestra aplicación, como por ejemplo actividades, servicios (desde el api 21) , BroadCast Receivers. Siempre que indiquemos el nombre del componente y este resida dentro de nuestra propia aplicación será considerado un intent explícito.

En la unidad “Introducción a android” si recuerdan la aplicación que tenía una actividad home y a través de un botón abría otra actividad que indicaba el estado de nuestra conexión a internet, este era un ejemplo de un intent explícito.

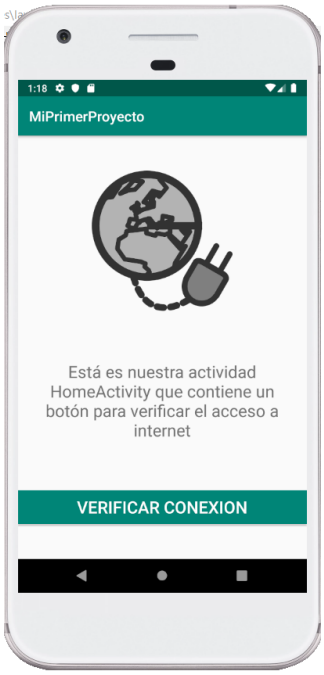

Imagen 6: Actividad A

Imagen 7: Actividad B

Ejercicio 8:

Aplicar acción al botón “Ver Respuesta” de nuestro fragmento a través de un intent implícito que nos permita ver los datos de nuestra api de opentdb en el navegador del teléfono.

1. En nuestra clase PreguntaFragmento, declararemos una nueva variable de clase Button.

```
private Button btnVerRespuesta;
```

2. En nuestro método initializeViews() , asociamos nuestra variable de clase btnVerRespuesta con la vista Button presente en nuestro archivo layout “fragment_pregunta.xml”.

```
btnVerRespuesta = view.findViewById(R.id.btnConsultaRespuesta);
```

3. En nuestro método onCreateView del mismo fragmento declaramos un método de evento de acción del botón para que cuando este sea presionado abra el navegador en nuestro teléfono y nos muestre los datos de la api que construye nuestras preguntas. Para este fin, usaremos la clase Uri para utilizar su método parse() con el parámetro de nuestra dirección web de opentdb.com y declaramos un Intent con los parámetros ACTION_VIEW y la uri de nuestra web.

```
btnVerRespuesta.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Uri uri = Uri.parse("https://opentdb.com/api.php?amount=10&category=15&difficulty=easy");
        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        startActivity(intent);
    }
});
```

4. El resultado al seleccionar el botón “Ver respuesta” es el que nos muestra la imagen siguiente:

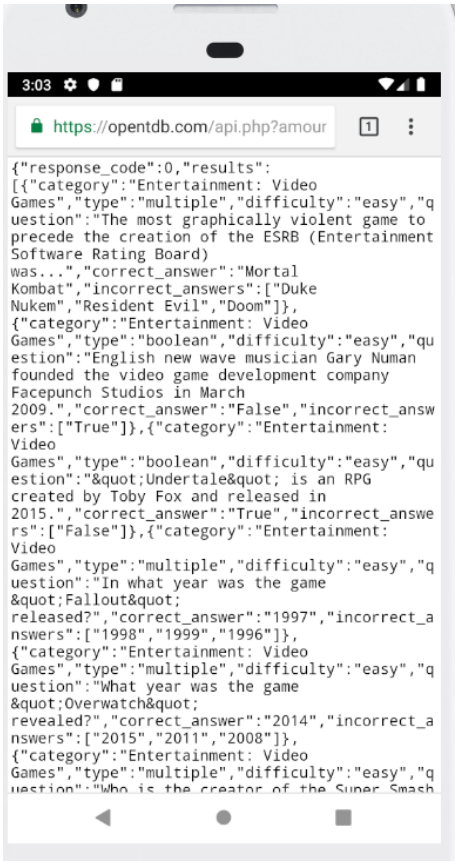

Imagen 8: Resultado Ejercicio 8

En el siguiente capítulo realizaremos ejercicios que complementarán el conocimiento adquirido en este capítulo de “Intents, implícitos y explícitos”.

Datos en intents y manejo de resultados

Competencias:

- Conocer la estructura de un intent
- Crear intents con resultados
- Pasar datos entre intents
- Pasar datos como resultados

Introducción

En el contexto del capítulo anterior daremos continuación al desarrollo de intents profundizando el envío de datos y su lectura.

En los próximos ejercicios haremos énfasis en la creación de una nueva clase que reciba datos recopilados de todas las respuestas desde la actividad anterior y de lectura de esta información para ser mostrada al usuario, además de indicar una nota simbólica no calculada para ser enviada a través de un intent implícito hacia aquellos programas que tengamos instalados para compartir.

Datos - Estructura de un Intent

La estructura de un intent se compone principalmente de dos atributos:

- action: la declaración de la acción que representa el intent, como por ejemplo ACTION_VIEW para abrir una web en un navegador o ACTION_SEND para enviar un mensaje.
- data: representa toda la información necesaria que requiera la acción para cumplir su operación.

Un intent tiene atributos adicionales que pueden ser declarados para complementar su funcionamiento entre los que destacan:

- category: entregando información adicional de la acción a ejecutarse. Un ejemplo sería el parámetro CATEGORY_LAUNCHER que indica que el resultado de la acción debe aparecer en el arranque en lo más alto de la jerarquía de componentes.
- type: este parámetro se aplica para indicar el MIME type de los datos, es decir, según convenciones web, esto se refiere a la indicación de que nuestros datos son por ejemplo, del tipo text/html, text/plain o video/mp4, para que el receptor del intent interprete el tipo de contenido que está recibiendo.
- component: como ya lo hemos indicado anteriormente, este parámetro hace la principal diferencia entre un intent implícito y un intent explícito. En el parámetro component indicamos el nombre del componente a llamar.
- extras: este parámetro se usa regularmente y nos sirve para enviar información extra a un intent, como por ejemplo una lista de datos. En sí, extras es un Bundle android, y uno de sus métodos más usados es el putExtra().

Ejercicio 9:

Agregar una segunda actividad que sea invocada desde nuestro último fragmento, habilitando este último un botón que indique “Finalizar” y accionando el intent para la segunda actividad. El estilo de esta actividad debe conter un imageview con el logo desafío latam, un textview titulo con la cadena ¡FELICIDADES! HAS TERMINADO TU NOTA ES, y un textview con un número simbólico de nota obtenida por las respuestas.

1. Creamos un nuevo layout de nombre “finish_activity.xml” con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <ImageView
        android:id="@+id/imagenLogoDesafioLatam"
        android:layout_width="100dp"
        android:layout_height="80dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/labelTitle"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:contentDescription="imagen camara"
        android:src="@drawable/desafiolatam"
        />
    <TextView
        android:id="@+id/labelTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="¡FELICIDADES! HAS TERMINADO TU NOTA ES"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:textSize="16dp"
        android:textStyle="bold"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/labelTitle"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginTop="20dp"
        android:textSize="46dp"
        android:textColor="@color/colorAccent"/>
</android.support.constraint.ConstraintLayout>
```

2. En nuestra carpeta cl.desafiolatam.pruebadinamica creamos una nueva clase actividad llamada “FinishActivity.java” extendiendo de AppCompatActivity y declarando su método onCreate() indicando que su layout es el que hemos creado anteriormente.

```
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;

public class FinishActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //INDICAMOS NUESTRO LAYOUT PARA LA ACTIVIDAD
        setContentView(R.layout.finish_activity);
       }
}
```

3. En nuestro archivo manifest.xml agregamos la nueva actividad:

```
<activity android:name=".FinishActivity">
    <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    </intent-filter>
</activity>
```

4. En el archivo layout “fragment_pregunta.xml” agregaremos un nuevo botón llamado “Finalizar”, que conviva al lado derecho del actual botón “Ver Respuesta” y que no sea visible. Este se hará visible hasta cuando tengamos en pantalla el último fragmento o la última pregunta.

```
<Button
    android:id="@+id/btnFin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="25dp"
    app:layout_constraintTop_toBottomOf="@id/radioGrupoRespuestas"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintLeft_toRightOf="@+id/btnConsultaRespuesta"
    android:text="FINALIZAR"
    android:background="@color/colorAccent"
    android:textColor="#FFFFFF"
    android:paddingHorizontal="20dp"
    android:visibility="gone"
    />
```

Para lograr la correcta posición necesitaremos editar la vista del botón “Ver Respuesta”, específicamente los siguientes atributos:

```
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toLeftOf="@+id/btnFin"
```

5. Para saber si es el último fragmento o no desde nuestro “FragmentoPregunta” debemos agregar un nuevo parámetro a nuestro método constructor que indique el total de preguntas.

```
public static PreguntaFragment newInstance(int numeroPregunta,
                                           String pregunta,
                                           String categoria,
                                           String respuestaCorrecta,
                                           ArrayList<String> respuestasIncorrectas,
                                           int totalPreguntas) {
    PreguntaFragment fragment = new PreguntaFragment();
    Bundle arguments = new Bundle();
    arguments.putInt("ID_FRAGMENTO", numeroPregunta);
    arguments.putString("NUMERO_PREGUNTA", "PREGUNTA "+numeroPregunta);
    arguments.putString("PREGUNTA", pregunta);
    arguments.putString("CATEGORIA", categoria);
    arguments.putString("RESPUESTA_CORRECTA", respuestaCorrecta);
    arguments.putStringArrayList("RESPUESTAS_INCORRECTAS", respuestasIncorrectas);
    arguments.putInt("TOTAL_PREGUNTAS", totalPreguntas);
    fragment.setArguments(arguments);

    return fragment;
}
```

6. Desde nuestra MainActivity declaramos el nuevo parámetro totalPreguntas en nuestro método getFragmentosDinamicos de la siguiente manera:

```
private List<Fragment> getFragmentosDinamicos(PreguntasLista preguntas){

    List<Fragment> fragmentosLista = new ArrayList<>();
    int contadorPregunta = 1;

    for(Pregunta pregunta: preguntas.getResults()) {
        fragmentosLista.add(
                PreguntaFragment
                        .newInstance(
                                contadorPregunta,
                                pregunta.getQuestion(),
                                pregunta.getCategory(),
                                pregunta.getCorrect_answer(),
                                pregunta.getIncorrect_answers(),
                                preguntas.getResults().size()
                        )
        );
        contadorPregunta++;
    }
    return fragmentosLista;
}
```

7. Desde nuestro método onCreateView() del fragmento capturamos el argumento recién creado en nuestro constructor newInstance.

```
final int totalPreguntas =
Objects.requireNonNull(getArguments()).getInt("TOTAL_PREGUNTAS");
```

8. En nuestra clase FragmentoPregunta, debemos declarar e iniciar nuestro nuevo botón como lo hemos hecho anteriormente con otras vistas. En el método onCreateView() antes de que finalice, debemos agregar una lógica que identifique que estamos ante el último fragmento o pregunta, y en base a esto, habilitar la vista del botón “Finalizar”, agregando además su acción OnClickListener, el cual hará un intent explícito de abrir una nueva actividad llamada “FinishActivity”.

```
private Button btnVerRespuesta, btnFin;
```

```
btnFin = view.findViewById(R.id.btnFin);
```

```
if(idFragmento +1 > totalPreguntas){
   btnFin.setVisibility(View.VISIBLE);
   btnFin.setOnClickListener(new View.OnClickListener(){
      @Override
       public void onClick(View v){
          Intent intentFinPreguntas = new
Intent(getActivity().getApplication().getBaseContext().getApplicationContext(), FinishActivity.class);
          intentFinPreguntas.putExtra("TOTAL_PREGUNTAS", totalPreguntas);
          intentFinPreguntas.putExtra("NOTA", 9);
          startActivity(intentFinPreguntas);
       }
   });
}
```

9. Finalmente declaramos la variable de clase TextView nota en nuestra actividad “FinishActivity” y declaramos su método onCreate(), con el respectivo layout, vistas y capturando los extras bundle que enviamos desde nuestro último fragmento para asignar su valor al textview nota.

```
package cl.desafiolatam.pruebadinamica;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.widget.TextView;
import java.util.Objects;

public class FinishActivity extends AppCompatActivity {
  private TextView notaFija;

  @Override
  protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //INDICAMOS NUESTRO LAYOUT PARA LA ACTIVIDAD
    setContentView(R.layout.finish_activity);
    //INICIAMOS NUESTRA VISTA
     initializeViews();

    //CAPTURAMOS DATOS ENVIADOS POR LA ACTIVIDAD ANTERIOR
    Bundle datosMainActivity = getIntent().getExtras();
    final int nota = Objects.requireNonNull(datosMainActivity).getInt("NOTA");
    //ASIGNAMOS VALOR A NUESTRA VISTA NOTA
    notaFija.setText(String.valueOf(nota));
  }

  private void initializeViews(){
    notaFija = findViewById(R.id.nota);
  }
}
```

10. El resultado de esta acción se demuestra en las siguientes imágenes consecutivas:

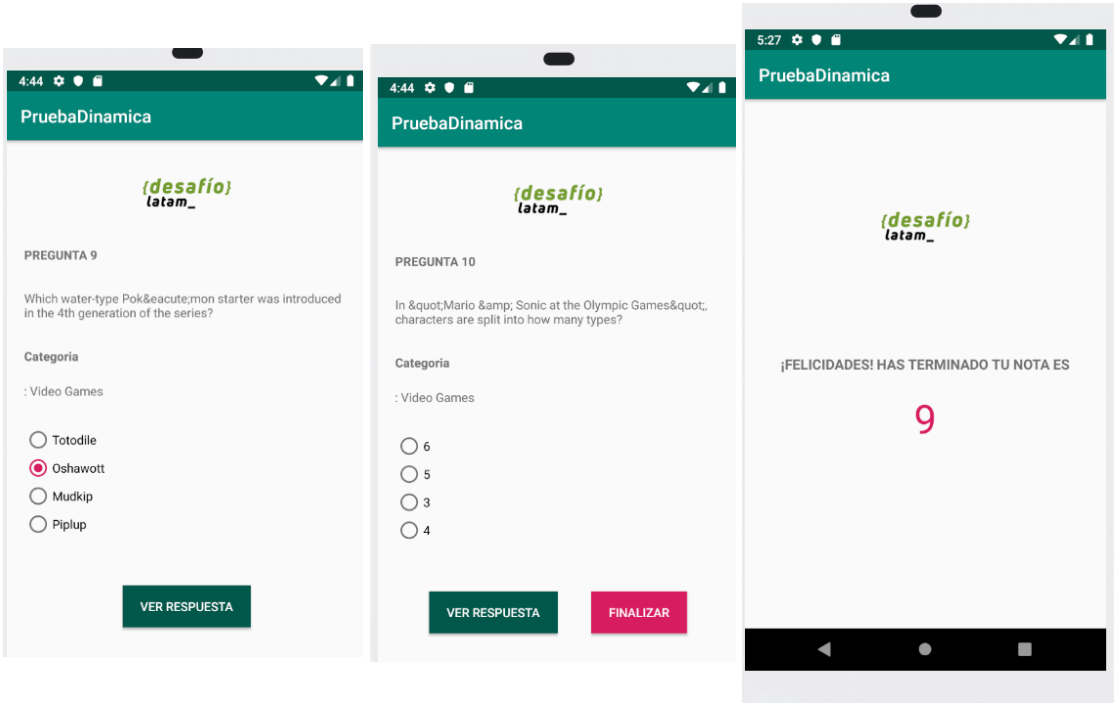

Imagen 9: Resultado Ejercicio 9

Ejercicio 10:

Agregar un botón en la segunda actividad que realice un intent implícito ACTION_SEND compartir, enviando la nota final a través de las aplicaciones disponibles en el teléfono.

1. Descargamos la imagen “share.png” etiquetada para reutilización, desde la siguiente url: https://cdn.pixabay.com/photo/2015/10/05/10/02/share-972389_960_720.png y la agregamos a nuestra carpeta drawable del proyecto.

2. A continuación en nuestro archivo layout “finish_activity.xml” agregamos un imageview al final de viewgroup, centrado, con la imagen descargada asociada.

```
<ImageView
   android:id="@+id/imgShareWhatsApp"
   android:layout_width="60dp"
   android:layout_height="60dp"
   android:onClick="shareWithWhatsApp"
   app:layout_constraintTop_toBottomOf="@+id/nota"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   android:src="@drawable/share"
   />
```

3. A continuación agregaremos un nuevo método en nuestra clase de actividad “FinishActivity” con el mismo nombre que tiene la declaración en la vista ImageView onClick. Este método se encargará de ejecutar la acción de compartir a través de un intent implícito de la siguiente manera:

```
public void shareWithWhatsApp(View v){
   Intent sendIntent = new Intent();
   sendIntent.setAction(Intent.ACTION_SEND);
   sendIntent.putExtra(Intent.EXTRA_TEXT, "¡Hola! te comparto mi niota obtenida hoy: " +
notaFija.getText().toString());
   sendIntent.setType("text/plain");
   //sendIntent.setPackage("com.whatsapp");
   startActivity(sendIntent);
}
```

4. Nótese del código anterior que está comentada una línea, esta indica que la aplicación a compartir sería whatsapp, sin embargo, para realizar la prueba correctamente desde nuestro emulador (ya que puede no tenga whatsapp instalado) permitiremos a la aplicación compartir nuestro mensaje a través de cualquier aplicación disponible para ello.

5. El resultado debería ser el siguiente, al hacer click sobre la imagen de compartir, el sistema de invitarte a abrir aplicaciones con las cuales puedas compartir tu mensaje.

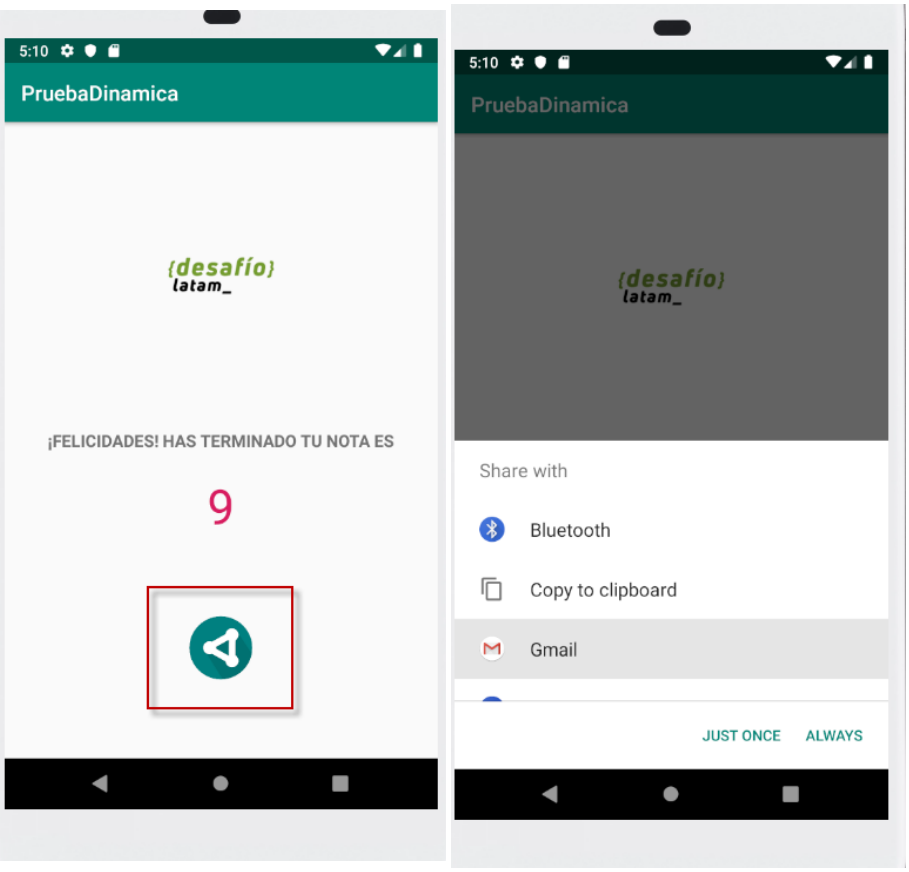

Imagen 10: Compartir a travpes d eun intent implícito

Entre las opciones que nos muestra un emulador en android studio, podemos escoger SMS para verificar que nuestro contenido está siendo enviado como lo muestra la siguiente imagen.

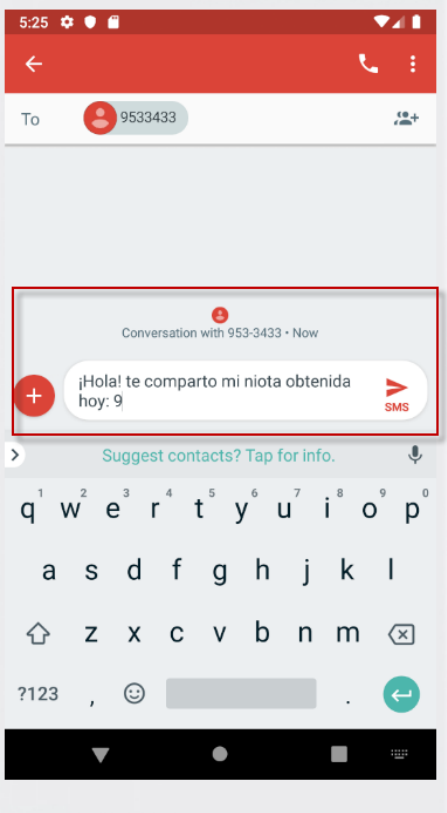

Imagen 11: Mensaje intent en SMS

Resultados en una actividad

El inicio de otra actividad no siempre tiene que ser en un único sentido. También puedes iniciar otra actividad y recibir un resultado. Para recibir un resultado, puedes invocar el método startActivityForResult() (en lugar de a startActivity()).

Una aplicación android, puede iniciar un intent hacia el correo electrónico y recibir la un resultado de la acción, por ejemplo. La actividad que responde debe estar preparada para mostrar un resultado. Cuando es así, envía el resultado en forma de otro objeto Intent.

La actividad que finalmente hará lectura de los resultados, ls recibe en el método onActivityResult().

Como ya hemos visto anteriormente a lo largo de la unidad, cuando se trata de fragmentos, podemos construir una interfaz que envíe los datos a nuestra actividad residente para que esta pueda realizar cualquier acción en ese contexto.

Ejercicio 11:

Resultados del intent compartir, lectura y envío de resultados a la actividad inicial.

1. En nuestra clase de actividad “FinishActivity”, editamos el método shareWithWhatsApp(), eliminando la siguiente línea:

```
startActivity(sendIntent);
```

2. En el mismo método shareWithWhatsApp agregamos dos instrucciones que nos permiten obtener los resultados y comenzar el intent de compartir con un request code 2 por defecto:

```
setResult(Activity.RESULT_OK,sendIntent);
startActivityForResult(sendIntent, 2);
```

3. En nuestra clase de actividad “FinishActivity”, agregaremos un nuevo método del ciclo de vida, el onActivityResult(), al final de dicha clase, donde capturaremos el resultado del intent compartir validando que el requestCode sea el 2.

4. Teniendo esto, a continuación crearemos un nuevo intent de retorno con un mensaje asociado y un Activity.RESULT_OK’, para ser interpretado desde nuestra clase de actividad “MainActivity”, no sin antes cerrar la actividad en curso “FinishActivity” con la instrucción finish(), de la siguiente forma:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  Log.d(TAG,"RequestCode: "+requestCode);

  if(requestCode == 2){
     Intent returnIntent = new Intent();
     returnIntent.putExtra("result","El usuario ha terminado con éxito, la nota resultante fue:
"+notaFija.getText().toString());
     setResult(Activity.RESULT_OK,returnIntent);
     finish();
  }
}
```

5. Para terminar, en nuestra clase de actividad “MainActivity” agregaremos el método del ciclo de vida onActivityResult(), validando un resultCode==’OK’ y mostrando a través de un mensaje Toast nuestro mensaje “result”.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

  if(resultCode==RESULT_OK){
     Toast.makeText(getApplicationContext(), "Hemos vuelto a las preguntas,
"+data.getStringExtra("result"), Toast.LENGTH_LONG).show();
     Log.d(TAG, "MainActivity "+data.getStringExtra("result"));
  }}
```