GIT

Comandi

- git status

> Mostra tutti i file non aggiornati con la repo remota o locale

- git checkout -b NAME

> Crea un branch di nome NAME e si sposta in esso

- git log --oneline

> Printa solo i messaggi di tutti i commit

- git init --bare NAME

> Crea una repository remota

- git remote add origin URL_REPOSITORY

> Collega la repo remota con la repo attuale

- git merge origin/master

> Unisce le due repo

- git fetch origin BRANCH

> Scarica tutti gli oggetti e i riferimenti da un'altra repository che sara' quella remota

- git pull

> Esegue i due comandi precedenti insieme

- git push -u origin/master

> Aggiorna la repo indicata

- git rm NOME

> leva il file dall'index

- git push -u origin: BRANCH

> elimina il branch remoto

- git branch -D BRANCH

> elimina il branch locale

ANDROID

Definizioni

- Activity: L'Activity è una singola cosa che l'utente può fare e con cui può interagire. Possono essere finestre fullscreen o fluttuanti. Utilizzano il metodo onCreate(Bundle) e onPause()

```
public class TestActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
       //DO THING
    }
}
```

- Fragment: Un Fragment è un pezzo di UI di un'applicazione che può essere piazzato in un'Activity. Il FragmentManager gestisce le interazioni tra i frammenti. Ogni frammento può essere usato con una sola Activity e ne condivide il ciclo di vita

```
//CREO IL FRAGMENT
public class DetailFragment extends Fragment
{

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle
savedInstanceState)
    {
        View view = inflater.inflate(R.layout.detail,
                container, false);
        return view;
    }

    public void setText(String txt)
    {
        TextView view = (TextView) getView().findViewById(R.id.detailsText);
        view.setText(txt);
    }
}

//NELL'ACTIVITY LO SETTA
public class MainActivity extends AppCompatActivity implements
MyListFragment.OnItemSelectedListener
{

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

```
    }

    @Override
    public void onItemSelected(String txt)
    {
        DetailFragment fragment = (DetailFragment) getFragmentManager()
                .findFragmentById(R.id.detailFragment);
        fragment.setText(txt);
    }
}
```

- Dialog: Finestre fluttuanti

```
public class MyDialogFragment extends DialogFragment {

    int style;
    int theme;

    // factory method
    static MyDialogFragment newInstance(int style, int theme)
    {
       //DO THINGS
        return f;
    }

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState)
    {
        //DO THINGS
    }
}

public class MainActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view)
     {
         //CREA DIALOG E CHIAMALO
```

```
    }
}
```

- LayoutInflater: Inizializza un layout XML nel suo corrispettivo View Object. Si ottiene con getLayoutInflater()

```
void TestInflater()
{
    LayoutInflater inflater = getLayoutInflater();
    View myLayout = inflater.inflate(R.layout.my_layout, mainLayout, false);
}
```

- Layout: Un Layout definisce una struttura nell'UI di un'App, come un'Activity. Tutti gli elementi nel Layout sono costruiti seguendo la gerarchia di View e ViewGroup

```
public class MainActivity extends AppCompatActivity
{

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.linear03); //SETTI IL LAYOUT XML
    }
}
```

- ViewGroup: Insieme di View e altri ViewGroup
- View: E' un componente che definisce un blocco basico contentente altri componenti.
- Adapter: E' un ponte tra un AdapterView e i dati relativi a quella View. Può essere reso custom creando una classe che estende ArrayAdapter. per il costruttore sono necessari contesto e un ID di UI con cui fare il super dopo. Deve implementare il metodo getView che restituisce una View e prende in input una posizione, una View e un ViewGroup (il parente della view). All'interno vanno inseriti nella UI i valori e restituita la view modificata. Sono opzionali i metodi getItem, getItemId ecc

```
void TestAdapter()
{
    GridView sampleView = (GridView)
linearLayout.findViewById(R.id.sample_layout);
    sampleView.setAdapter(new SampleAdapter());
}

public class ListAdapter extends ArrayAdapter<Item>
{
    public ListAdapter(Context context, int textViewResourceId)
```

```
    {
        super(context, textViewResourceId);
    }

    public ListAdapter(Context context, int resource, List<Item> items)
    {
        super(context, resource, items);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        View v = convertView;
        if (v == null)
        {
            LayoutInflater vi;
            vi = LayoutInflater.from(getContext());
            v = vi.inflate(R.layout.itemlistrow, null);
        }
        Item p = getItem(position);
        if (p != null)
        {
            TextView tt1 = (TextView) v.findViewById(R.id.id);
            TextView tt2 = (TextView) v.findViewById(R.id.categoryId);
            TextView tt3 = (TextView) v.findViewById(R.id.description);
            if (tt1 != null)
            {
                tt1.setText(p.getId());
            }

            if (tt2 != null)
            {
                tt2.setText(p.getCategory().getId());
            }

            if (tt3 != null)
            {
                tt3.setText(p.getDescription());
            }
        }
        return v;
    }
}
```

- ListView: E' un ViewGroup che mostra una lista di oggetti scrollabili

```
public class MainActivity extends AppCompatActivity
{

    private ArrayAdapter<String> adapter;
    private ArrayList<String> products;
```

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

        initializeProducts();

        ListView mListView = findViewById(R.id.my_list_view);
        adapter = new ArrayAdapter<>(this, android.R.layout.test_list_item,
products);
        mListView.setAdapter(adapter);
    }

    public void populate(View view)
    {
        adapter.clear();
        initializeProducts();
        adapter.addAll(products);
        adapter.notifyDataSetChanged();
    }

    private void initializeProducts()
    {
        products = new ArrayList<>(Arrays.asList(
                "gioppini",
                "jambonetti",
                "patatine sfizione",
                "tarallini",
                "gallette",
                "frollini plus",
                "cioccolini",
                "secchini",
                "grissinini",
                "patasplash",
                "majopatas",
                "crocchette al sesamo",
                "crocchette alla pancetta",
                "biscotti al miglio e avena"
        ));
    }
}
```

- Intent: Un intent è una descrizione astratta di un'operazione che deve essere performata, ad esempio con startActivity

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

```
    Button btn = findViewById(R.id.button);

    btn.setOnClickListener((view)-> {

        Intent i = new Intent(this, ActivityTwo.class);

        // put extra
        i.putExtra("str1", "Message 1");
        i.putExtra("n1", 111);

        // put extras
        Bundle extras = new Bundle();
        extras.putString("str2", "Message 2");
        extras.putInt("n2", 222);
        i.putExtras(extras);

        // start activity for result
        startActivityForResult(i, ACTIVITY_TWO);
    });
}
```

- Bundle: E' una mappa che usa stringhe come chiavi e come valori oggetti che implementano Cloneable e Parcelable

```
Bundle TestBundle()
{
    Bundle simple_bundle=new Bundle();
    simple_bundle.putString("item1","value1");
    return simple_bundle;
}
```

- BroadcastReceiver: Classe che gestisce gli intent in broadcast settati con sendBroadcast(Intent)
- Retrofit: Retrofit è una libreria REST per la comunicazione internet. La dichiarazione di azioni retrofit avviene tramite l'annotazione a parti. Ad esempio:

```
@Multipart //insieme di parti
@PUT("user/photo") //Comando
Call<User> updateUser( //metodo per richiamare l'oggetto modificato
  @Part("photo") RequestBody photo, //parte aggiunta
  @Part("description") RequestBody description //parte aggiunta
);
```

per quanto riguarda invece la classe vera e propria per usare retrofit bisogna inizializzare il client

```
public static Retrofit getClient(String baseUrl)
{
    if (retrofit==null)
    {
        retrofit = new Retrofit.Builder()
                .baseUrl(baseUrl)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
    }
    return retrofit;
}

Retrofit c = getClient("URL");
FileUploadService fus = c.create(FileUploadService.class);
//File Upload Service è una classe tipo quella di sopra con tutte le parts. Devi creartela. non estende nulla, ha solo quella roba dentro
```

inoltre è anche necessario creare le parti manualmente per poi inviarle (in quesot caso immagini)

```
private void getParts(File file, String descriptionString)
{
    // create RequestBody instance from file
    RequestBody requestFile =
            RequestBody.create
            (
                    MediaType.parse("image/*"),
                    file
            );
    // MultipartBody.Part is used to send also the actual file name
    MultipartBody.Part body =
            MultipartBody.Part.createFormData("image", file.getName(),
requestFile);
    RequestBody description = RequestBody.create(
            // multipart/form-data
            MultipartBody.FORM, descriptionString);

    // finally, execute the request
    Call<ResponseBody> call = mService.upload(description, body);
    call.enqueue(new Callback<ResponseBody>()
    {
        @Override
```

```
        public void onResponse(Call<ResponseBody> call,Response<ResponseBody>
response)
        {

            Log.v("Upload", "success");
            Toast.makeText(MainActivity.this, getString(R.string.success),
Toast.LENGTH_LONG).show();
        }
        @Override
        public void onFailure(Call<ResponseBody> call, Throwable t)
        {
            Log.e("Upload error:", t.getMessage());
            Toast.makeText(MainActivity.this, getString(R.string.failure),
Toast.LENGTH_LONG).show();
        }
    });
}
```

- Ciclo di vita dell'Activity: L'Activity ha 6 callbacks:
- onCreate(): Chiamato come primo metodo alla creazione

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

- onStart(): Chiamato subito dopo l'onCreate
- onResume(): Chiamato dopo il resume dell'activity

```
@Override
public void onResume()
{
    super.onResume();
}
```

- onPause(): Chiamato quando si mette in pausa l'activity

```
@Override
public void onPause()
{
    super.onPause();
}
```

- onStop(): chiamato quando si esce dall'activity

```
@Override
public void onStop()
{
    super.onStop();
}
```

- onDestroy(): chiamato durante la distruzione dell'activity
- Comunicazione tra Activity:
- startActivity

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

- startActivityForResult

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == PICK_CONTACT_REQUEST)
    {
        if (resultCode == RESULT_OK)
        {
            startActivity(new Intent(Intent.ACTION_VIEW, data));
        }
    }
}
```

- ViewHolder:
- Comunicazione tra Fragment: la comunicazione tra Fragment avviene sempre passando per l'Activity padre. per fare bisogna fare diverse cose: un'interfaccia che l'activity implementerà il metodod onAttach nel fragment (che riceverà un'activity e se la casterà all'interfaccia per poi salvarla) chiamare il metodo dell'interfaccia dal fragment sfruttando l'activity salvata. All'interno di questo metodo gestirsi con getSupportFragmentManager().findFragmentById ecc ecc. più info qui https://developer.android.com/training/basics/fragments/communicating
- DB Helper: Il DB helper è una classe che estende SQLiteOpenHelper e implementa i seguenti metodi:
- onCreate(SQLiteDatabase database) Si esegue alla creazione
- onUpgrade(SQLiteDatabse db, int old, int new) Aggiorna la versione del DB E'consigliata inoltre una Stringa iniziale per il nome del database e una per la versione. In questo caso facendo il super del costruttore (ricevendo il Context dell'activity) si potranno inserire come dati

```
public MySQLiteHelper(Context context)
{
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase database)
{
    database.execSQL(DATABASE_CREATE);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
    Log.w(MySQLiteHelper.class.getName(),
            "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");

    db.execSQL("DROP TABLE IF EXISTS " + TABLE_PRODUCTS);
    onCreate(db);
}
```

- Recycler View