# TP1

* Télécharger GitHub Desktop et Git pour Windows.
* GitHub Desktop -> File -> Clone repository -> coller le lien copié depuis le github <https://github.com/gxfab/LO52_A2019.git>
* $git config –global user.name “xxx”;

$git config –global user.email [xxx@example.com](mailto:xxx@example.com)

(Linux)

* GitHub desktop -> Branche -> Name : FGurlsDev -> Create branch based on master
* GitHub desktop -> Current branch -> FGurlsDev
* Créer directement un répertoire dans le répertoire cloné s’appelle LO52\_2019\_FGurlsDev, modifier le fichier README.txt.
* Télécharger et installer AndroidStudio.
  + Java (jdk, jre)
  + SDK
* Android Studio -> Start a new Android Studio project -> Configurer les informations du projet -> Empty Activity (MainActivity.java)

Rajouter un bouton dans le layout principal (activity\_main.xml).

Configurer l’action de bouton (MainActivity.java, setOnClickListener)

Créer le deuxième activité (SecondActivity.java)

Rajouter un label dans le layout secondaire (activity\_second.xml)

* + Remarques :
    - Pour utiliser l’activité existée pour activer la nouvelle activité, la nouvelle activité n’est pas encore créée, donc nous devons utiliser newActivity.class mais pas newActivity.this.
    - ‘This’ est une référence d’un membre de l’objet actuel à partir d’une méthode d’instance ou d’un constructeur.
  + Problème : Android Studio n’arrive pas à retrouver le divice HUAWEI
    - Raison et Solution : drive de huawei ne peut pas être utilisé par Android Studio, donc télécharger Google USB Driver dans SDK Manager – Google USB Driver.
  + Pour changer la nouvelle activité, utiliser la classe Intent.
* Copier le répertoire src dans le repertoire LO52\_A2019/LO52\_2019\_FGurlsDev/Part 1

(TP2 est supprimé)

# TP3

* Merger la branche SnakeTeacher dans la branche FGurlsDev :
  1. GitHub Desktop -> Current Branch -> **FGurlsDev**
  2. Branch -> Merge into current branch -> Merge **SnakeTeacher** into **FGurlsDev**
* Implémentation de la libusb :
  1. Fichiers sources : core.c, descriptor.c, io.c, sync.c, os/darwin\_usb.c, os/linux\_usbfs.c

Fichiers headers : libusb.h, libusbi.h, os/darwin\_usb.h, os/linux\_usbfs.h

* 1. Modifier le fichier Rsc-libusb/libusb-1.0.3/libusb/Android.mk (MakeFile pour le composant libusb) :

LOCAL\_PATH:=$(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES:= \

core.c \

descriptor.c \

io.c \

sync.c \

os/darwin\_usb.c \

os/linux\_usbfs.c

LOCAL\_C\_INCLUDES+= \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/os

LOCAL\_MODULE:=libusb

include $(BUILD\_SHARED\_LIBRARY)

* 1. Solution pour l’erreur TIMESPEC\_TO\_TIMEVAL - Rajouter les codes suivant dans le fichier io.c :

#define TIMESPEC\_TO\_TIMEVAL(tv, ts) do { \

(tv)->tv\_sec = (ts)->tv\_sec; \

(tv)->tv\_usec = (ts)->tv\_nsec / 1000; \

} while (0)

* 1. Solution pour l’erreur library "libusb.so" not in prelink map – Rajouter la ligne suivante dans le fichier Android.mk : LOCAL\_PRELINK\_MODULE:=false
* Implémentation d’un nouveau produit Android :
  1. Renommer le fichier Rsc-libusb/device/TP3\_A\_VOUS\_JOUEZ.mk comme lo52\_FGurlsDev.mk
  2. Modifier ce fichier :
     + Configurer les informations simples de produit :

PRODUCT\_NAME:=lo52\_FGurlsDev

PRODUCT\_DEVICE:=lo52\_FGurlsDev

PRODUCT\_BRAND:=lo52\_FGurlsDev

PRODUCT\_MODEL:=lo52\_FGurlsDev

* + - Il héritera du produit hikey de Linaro :

$(call inherite-product,device/Linaro/hikey/hikey.mk)

* + - Personnaliser les propriétés :

PRODUCT\_PROPERTY\_OVERRIDES+= \

ro.hw=lo52 \

net.dns1=8.8.8.8 \

net.dns2=4.4.4.4

* Surcharger le fichier sym\_keyboard\_delete.png
  1. Chercher “aosp sym\_keyboard\_delete.png” sur Google;
  2. Cliquer sur premier lien, le fichier peut être trouvé dans le projet platform\_frameworks\_base;
  3. Créer un répertoire **overlay** dans le répertoire **Rsc-libusb/device**;
  4. Créer les répertoire avec la même arborescence mirror comme ce que l’on a trouver dans le projet :

overlay/platform\_frameworks\_base/core/res/res/drawable-en-mdpi

* 1. Rajouter cette ligne de code dans le fichier lo52\_FGurlsDev.mk :

DEVICE\_PACKAGE\_OVERLAYS:=overlay

* Ajouter la libusb aux packages du produit : PRODUCT\_PACKAGES+=libusb
* Commit les modifications sur la branche FgurlsDev.

# TP4

### Configuration de l’environnement NDK

Android studio

Tools -> SDK Manager -> SDK Tools -> télécharger NDK, CMAKE, LLDB

File -> SDK Location -> Android NDK location : (default)

Copier le chemin par défaut, créer une nouvelle variable d’environnement « NDK\_ROOT », coller le chemin dans la champe « value », rajouter cette %NDK\_ROOT% dans Path.

Cmd -> ndk-build pour tester s’il est bien configuré.

### Création du projet

Créer un projet de type Native C++, choisir le langage Java et « Toolchain Default » comme la version de C++ Standard que nous voulons utiliser pour écrire les code sources de librairie JNI.

**Remarques** :

Il y a quelques fichiers et répertoires importants sont créés.

**src/main/cpp** : une répertoire qui contient

* CMakeLists.txt : un fichier .txt généré lors de la création du projet, il permet d’indiquer et d’ajouter les librairies JNIs dans le projet.
* native-lib.cpp : Le fichier .cpp qui contient les codes sources de librairie "native-lib" en langage C/C++.

**/app/build.gradle** : fichier de configuration qui gère la construction de projet, les lignes de codes suivantes permettent de lancer l’installation de librairie JNI :

externalNativeBuild {

cmake {

path "src/main/cpp/CMakeLists.txt"

version "3.10.2"

}

}

Créer une nouvelle classe NDKTools.java qui contien la déclaration de la librairie personnalisé et déplace les lignes de code de MainActivity.java suivants dans NDKTools.java :

static {  
 System.loadLibrary("native-lib");  
}

public native static String stringFromJNI();

Qui décrit :

* Le nom de librairie : native-lib, ce qui doit être la même avec ce qui est marqué dans le CMakeLists.txt :

add\_library( # Sets the name of the library.

native-lib

# Sets the library as a shared library.

SHARED

# Provides a relative path to your source file(s).

native-lib.cpp)

* Et les noms de fonction dans la librairie.

**Erreur apparaît** : “Cannot resolve corresponding JNI function Java\_fr\_utbm\_fgurlsdev\_tp4jni\_NDKTools\_stringFromJNI”

**Raison** : le nom de fonction JNI n’est pas correct ou il n’existe pas dans le code source.

**Solution** : Aller dans le fichier native-lib.cpp, modifier le nom de fonction comme le suivant : Java\_fr\_utbm\_fgurlsdev\_tp4jni\_NDKTools\_stringFromJNI, et modifier le type de son deuxième paramètre comme jclass.

Rajouter les fonctions suivantes :

public native static String sayHi(); // Celle-ci est pour tester  
public native static String readBtn(String numStr);  
public native static String writeBtn(String numStr);  
public native static String directionBtn(String direction);

Définir les fonctions dans le fichier native-lib.cpp avec les noms suivants :

Java\_fr\_utbm\_fgurlsdev\_tp4jni\_NDKTools\_sayHi

Java\_fr\_utbm\_fgurlsdev\_tp4jni\_NDKTools\_readBtn

Java\_fr\_utbm\_fgurlsdev\_tp4jni\_NDKTools\_writeBtn

Java\_fr\_utbm\_fgurlsdev\_tp4jni\_NDKTools\_directionBtn

//Définit sous le format : Java\_package\_Classe\_nomDeFonction

(Le détail est dans les fichiers sources)

Mettre en place l’interface d’application et les fonctionnalités bouton.

Comme le principe de fonction des boutons "UP", "DOWN", "LEFT", "RIGHT" est le même, on a décider de créer un nouveau objet DrectionBtnListener qui hérite l’interface OnClickListener, pour gérer l’action après on clique sur un de ces 4 boutons.