Android

2020/2021

Yousri Daldoul

Maître Assistant en Informatique

Faculté des Sciences de Monastir

yousri.daldoul@fsm.rnu.tn

Introduction

Android :

- Un OS basé sur Linux, pour les smatphones et les tablettes.
- Un OS open source
- Développé par Google
- Initialement conçu pour les smartphones et tablettes tactiles, puis s'est diversifié : les télévisions (Android TV), les voitures (Android Auto), les ordinateurs (Android-x86) et les smartwatch (Android Wear).
- Le système d'exploitation mobile le plus utilisé au monde

Parts de marché des OS mobiles en 2020 : (www.idc.com/promo/smartphone-market-share/os)

- Android 85%
- iOS : 15%
- Autres : 0%

Développement des applications Android :

- Développement avec Java sous « **Android Studio** » : génération du fichier exécutable « **.apk** »
- Développement avec JavaScript et Cordova : conversion en Java et génération du fichier exécutable « **.apk** »
- L'exécution de l'exécutable « **.apk** » nécessite la machine virtuelle Dalvik (pour les anciennes versions du OS) ou ART (pour les nouvelles versions du OS).

Télécharger une application Android :

- Google Play
- Market alternatifs (Amazon Appstore for Android, GetJar, F-Droid, Samsung Galaxy Apps, etc.)
- Téléchargement direct du fichier binaire (.apk) sur le smartphone

Google Play :

- Un magasin d'applications créé par Google (le 6 mars 2012)
- Une boutique d'applications pour le système d'exploitation Android
- En 2020, Google Play contient 3 000 000 applications, ce qui en fait le plus gros magasin d'applications au monde devant iOS et Windows Phone (https://www.appbrain.com/stats)

Les Versions d'Android :

Version Android	Date de sortie	API Level	Nom
Android 11	8 Septembre 2020	30	Android 11
Android 10	3 Septembre 2019	29	Android 10
Android 9.0	6 août 2018	28	Pie
Android 8.1	5 Décembre 2017	27	Oreo
Android 8.0	21 août 2017	26	Oreo
Android 7.1	4 Octobre 2016	25	Nougat
Android 7.0	Aout 2016	24	Nougat
Android 6.0	Aout 2015	23	Marshmallow
Android 5.1	Mars 2015	22	Lollipop
Android 5.0	Novembre 2014	21	Lollipop
Android 4.4W	Juin 2014	20	Kitkat Watch
Android 4.4	Octobre 2013	19	Kitkat
Android 4.3	Juillet 2013	18	Jelly Bean
Android 4.2	Novembre 2012	17	Jelly Bean
Android 4.1	Juin 2012	16	Jelly Bean
Android 4.0.3	Décembre 2011	15	Ice Cream Sandwich
Android 4.0	Octobre 2011	14	Ice Cream Sandwich
Android 3.2	Juin 2011	13	Honeycomb
Android 3.1	Mai 2011	12	Honeycomb
Android 3.0	Février 2011	11	Honeycomb
Android 2.3.3	Février 2011	10	Gingerbread
Android 2.3	Novembre 2010	9	Gingerbread
Android 2.2	Juin 2010	8	Froyo
Android 2.1	Janvier 2010	7	Eclair
Android 2.0.1	Décembre 2009	6	Eclair
Android 2.0	Novembre 2009	5	Eclair
Android 1.6	Septembre 2009	4	Donut
Android 1.5	Mai 2009	3	Cupcake
Android 1.1	Février 2009	2	Base
Android 1.0	Octobre 2008	1	Base

Remarque : Une application développée avec une API (exemple l'API 21), fonctionne correctement sur des systèmes d'exploitation utilisant des API plus récentes (exemple API 25) mais ne fonctionne pas correctement sur des systèmes d'exploitation utilisant des API plus anciennes (exemple API 15).

Pour développer des applications Android, il faut :

- Le « Software Development Kit » (SDK) qui contient les librairies (API) nécessaires pour développer des applications Android + les outils de compilation
- Un environnement de développement intégré (IDE : Integrated Development Environment) permettant d'écrire le code source et de générer l'application
  ⇨ **« Android Studio » (dernière version 4.1.2) : https://developer.android.com/studio**
- Un smartphone Android pour tester les applications réalisées. Il y a 2 possibilités

- On utilise un émulateur qui est un programme qui simule un Smartphone Android (certaines machines ne supportent pas la Virtualisation et ne peuvent pas utiliser l'émulateur)
- On utilise un Smartphone Android réel connecté par câble USB à l'ordinateur : Il faut activer l'option « USB debugging » à partir de « paramètres > options développeur »

Android studio contient les 2 outils suivants :

- Software Development Kit (SDK) Manager : permet de visualiser les différentes versions de SDK (un numéro API correspond à chaque version de SDK) disponibles et les télécharger. Accès possible à partir de « **Tools > Android > SDK Manager** »
- Android Virtual Device (AVD) Manager : gère les différents émulateurs. Il est possible de créer plusieurs émulateurs pour les différentes tailles de tablettes et de Smartphones. Accès possible à partir de « **Tools > Android > AVD Manager** »

Le langage Java :

- Permet de développer des applications bureautiques (Java Standard Edition : SE), des applications web (Java Entreprise Edition : EE), des applications mobiles (Android et Java Micro Edition), etc.
- L'exécutable tourne sur une machine virtuelle
- La gestion de la mémoire est facile : le garbage collector s'en occupe !

I. Langage Java

1) Programmation Orienté Objet

Java est un langage uniquement « **orienté objet** » : Les variables et les fonctions sont regroupées dans des classes. Dans le cas de Java, il n'est pas possible de déclarer une variable ou une fonction à l'extérieur d'une classe.

Exemple d'une classe :

```
public class Exemple {
    // Cette ligne est un commentaire car elle commence par //
    // variables et fonctions
}
```

Remarque : Une classe java doit être sauvegardée dans un fichier qui porte le même nom de la classe et l'extension « **.java** ». Donc la classe « **Exemple** » doit se trouver dans le fichier « **Exemple.java** ».

Compilation :

Un code Java doit être compilé pour obtenir un fichier exécutable : un fichier qui contient un code binaire compréhensible uniquement par la machine virtuelle

Code source : un ou plusieurs fichiers avec l'extension « **.java** »	→	**Byte code** : un ou plusieurs fichiers avec l'extension « **.class** »	→	**Exécutable** : un seul fichier compressé (contient des fichiers class, xml, images, etc.) avec l'extension « **.jar** »

L'exécution d'un programme consiste à exécuter ses instructions une par une, jusqu'à la fin des instructions. Le point d'entrée d'un programme est la fonction « **main** » :

```
public static void main(String[] args) {
    // code
}
```

Remarque :

- Il n'est pas possible d'exécuter un programme qui ne contient pas une fonction « **main** »
- Un fichier « **jar** » qui ne contient aucune fonction « **main** » est une librairie et non pas un exécutable.

Le programme exécutable minimal est :

```
public class Exemple {
    public static void main(String[] args) {
        System.out.println("bonjour");
    }
}
```

Convention de nommage

La convention de nommage est une façon d'appeler les variables, les classes, et les fonctions. Il faut essayer au maximum de la respecter. Cette convention est la suivante :

- tous vos noms de classes doivent commencer par une majuscule
- tous vos noms de variables doivent commencer par une minuscule
- si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation
- tout ceci sans accentuation !

2) Les variables

Une variable permet de stocker des informations de toute sorte en mémoire : des chiffres, des résultats de calcul, des tableaux, etc. Java est un langage fortement typé : toute variable doit avoir un type.

La déclaration d'une variable se fait comme suit :

```
type_variable nom_variable;
```

Ensuite, on initialise la variable en entrant une valeur :

```
nom_variable = valeur;
```

Il est possible de déclarer et d'initialiser une variable en une seule fois :

```
type_variable nom_variable1 = valeur1;
type_variable nom_variable2 = valeur2;
```

En Java, nous avons deux types de variables :

- Des variables de type simple ou « **primitif** » : **byte**, **short**, **int**, **long**, **float**, **double**, **char** et **boolean**
- Des variables de type complexe ou des « **objets** » : String, Vector, Button, etc.

Les variables de type numérique

Le type **byte** (1 octet) peut contenir les entiers entre -128 et +127	`byte temperature;` `temperature = 64;`
Le type **short** (2 octets) contient les entiers compris entre -32768 et +32767	`short vitesseMax;` `vitesseMax = 32000;`
Le type **int** (4 octets) contient des entiers.	`int temperatureSoleil;` `temperatureSoleil = 15600000;`
Le type **long** (8 octets) contient des très grandes valeurs d'entiers. Il faut ajouter un "L" à la fin du nombre si sa valeur dépasse la valeur maximale d'un entier.	`long anneeLumiere;` `anneeLumiere = 9460700000000000L;`

Le type **float** (4 octets) est utilisé pour les nombres avec une virgule flottante.
On utilise un point et non pas une virgule, le tout suivi de « **f** ».
	`float pi;`
	`pi = 3.141592653f;`
	`float nombre;`
	`nombre = 2.0f;`

Le type **double** (8 octets) ressemble à **float**, mais il peut contenir plus de chiffres derrière la virgule et n'a pas besoin d'un suffixe (on peut ajouter « **d** »).	`double division;` `division = 0.333333333333;`

Les variables de type booléen (boolean)

Le type **boolean** ne peut contenir que deux valeurs : **true** ou **false**, sans guillemets.	`boolean question;` `question = true;`

Les variables de type caractère (char)

Le type **char** contient un caractère stocké entre apostrophes (« ' ' »)	`char caractere;` `caractere = 'A';`

Les variables de type chaine de caractères (String)

Le type String permet de gérer les chaînes de caractères, c'est-à-dire le stockage de texte. Il s'agit d'une variable de type « **objet** ». String commence par une majuscule ! Son initialisation nécessite des guillemets doubles (« " " »).	`String phrase;`
	`phrase = "Une phrase";`
	
	`String chaine = new String();`
	`chaine = "Une chaîne de caractères";`
	
	`String string = "Une autre chaîne";`
	
	`String str = new String("une de plus !");`

Les opérateurs arithmétiques

- « + » : additionner deux variables numériques et concaténer des chaînes de caractères
- « - » : soustraire deux variables numériques
- « * » : multiplier deux variables numériques
- « / » : diviser deux variables numériques
- « % » : calculer le modulo (le reste) de la division entière de deux variables numériques

Quelques exemples de calcul

```
int nbre1, nbre2, nbre3;  //Déclaration des variables

nbre1 = 1 + 3;            //nbre1 vaut 4
nbre2 = 2 * 6;            //nbre2 vaut 12
nbre3 = nbre2 / nbre1;    //nbre3 vaut 3
nbre1 = 5 % 2;            //nbre1 vaut 1, car 5 = 2 * 2 + 1
nbre2 = 99 % 8;           //nbre2 vaut 3, car 99 = 8 * 12 + 3
nbre3 = 6 % 3;            //nbre3 vaut 0, car il n'y a pas de reste
```

```
int nbre1, nbre2, nbre3;    //Déclaration des variables
nbre1 = nbre2 = nbre3 = 0;  //Initialisation

nbre1 = nbre1 + 1;     //nbre1 = lui-même, donc 0 + 1 => nbre1 = 1
nbre1 = nbre1 + 1;     //nbre1 = 1 + 1 = 2

nbre2 = nbre1;         //nbre2 = nbre1 = 2
nbre2 = nbre2 * 6;     //nbre2 = 2 * 6 = 12

nbre3 = nbre2;         //nbre3 = nbre2 = 12
nbre3 = nbre3 / 4;     //nbre3 = 12 / 4 = 3

nbre1 = nbre1 - 1;     //nbre1 = 2 - 1 = 1
```

Les raccourcies :

```
// augmenter la valeur d'une variable par 1
nbre1 = nbre1 + 1;
nbre1 += 1;
nbre1++;
++nbre1;
```

```
// diminuer la valeur d'une variable par 1
nbre1 = nbre1 - 1;
nbre1 -= 1;
nbre1--;
--nbre1;
```

```
// multiplier la valeur d'une variable par 2
nbre1 = nbre1 * 2;
nbre1 *= 2;
```

```
// diviser la valeur d'une variable par 2
nbre1 = nbre1 / 2;
nbre1 /= 2;
```

Remarque : en faisant des opérations arithmétiques, il est possible d'avoir des pertes de données.
Exemple : « `int i = 3 / 2;` » (on obtient i = 1)

Les conversions (cast)

Les variables de type **double** contiennent plus d'informations que les variables de type **int**, et l'affectation d'un double à un entier peut causer la perte des données. Donc le compilateur génère une erreur lorsqu'il y a la possibilité de perte des données. Pour éviter ces erreurs, on fait la conversion des variables.

```
int i1 = 123;
float f1 = (float) i1;    // conversion (inutile car pas de perte)
                          // de int à float
double d1 = (double) i1; // conversion (inutile car pas de perte)
                          // de int à double

double d2 = 1.23;
double d3 = 2.985;
int i2 = (int) d2;     // conversion de double à int : i2 = 1
i2 = (int) d3;         // conversion de double à int : i2 = 2
```

3) Les classes

Une classe en Java permet de définir un ensemble de variables et de fonctions. Après la création d'une classe, on peut l'utiliser pour déclarer et initialiser des objets (un objet est une instance d'une classe : instanciation d'une classe = initialisation d'un objet d'une classe).

Les variables définies à l'intérieur d'une classe sont des variables **globales** (accessibles par toutes les méthodes de la classe, et même à l'extérieur de la classe si elles sont publiques). Les variables définies à l'intérieur d'une méthode sont des variables **locales** (accessibles à l'intérieur de la méthode uniquement).

Le nom d'une variable globale doit être unique dans toute la classe. Le nom d'une variable locale doit être unique dans la méthode. Il est possible qu'une variable locale porte le même nom qu'une variable globale (dans ce cas la variable locale est prioritaire dans la méthode) : pour utiliser la variable globale dans la méthode, on utilise le mot-clé « **this** » (pointeur sur la classe elle-même).

Création de la classe « Personne » :

```
public class Personne {
    public String nom;
    public String prenom;
    public int age;
}
```

Déclaration et instanciation (initialisation d'un objet) d'une classe avec le mot-clé « **new** » :

```
Personne personne;
personne = new Personne();
personne.nom = "Abid";
personne.prenom = "Amine";
personne.age = 21;
```

Le constructeur d'une classe est une méthode qui se charge de créer et d'initialiser un objet. Chaque classe a un constructeur par défaut. Il est possible de redéfinir le constructeur par défaut, et de créer différents constructeurs pour la même classe.

Le constructeur est une méthode qui ne retourne rien. Tous les constructeurs de la classe doivent porter le nom exact de la classe. Les constructeurs de la même classe doivent avoir des paramètres différents.

```
public class Personne {
    public String nom;
    public String prenom;
    public int age;

    public Personne() {
        nom = "";
        prenom = "";
        age = 0;
    }
```

```
    public Personne(String nom, String prenom, int age) {
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
    }
}
```

Le code suivant est **faux** car les deux constructeurs de la classe « **Point** » ont les mêmes paramètres :

```
// le code suivant est faux !!!
public class Point {
    public Point(int a, int b) {}
    public Point(int x, int y) {}
}
```

Déclaration et initialisation d'un objet :

```
Personne personne1, personne2;
personne1 = new Personne();
personne1.nom = "Abid";
personne1.prenom = "Amine";
personne1.age = 21;

personne2 = new Personne("Dridi", "Ikram", 20);
```

Il est possible d'initialiser un objet avec la valeur « **null** », mais il n'est pas possible de l'utiliser avant de l'initialiser avec une valeur valide !

```
Personne p1 = null, p2 = null;
p1 = new Personne("Abid ", " Amine ", 21); //p1 a une valeur valide
p2.nom = "Dridi"; //p2 est null (inutilisable) => erreur d'exécution
```

Les paquetages (package) :

Soit la classe « **Vector** » qui se trouve dans le paquetage « **java.util** » (dossier java/util) de la librairie standard (J2SE). Pour déclarer et initialiser un objet de cette classe, il faut importer la classe en utilisant le mot-clé « **import** ». Exemple :

```
import java.util.Vector;

public class Historique {
    Vector vector = new Vector();
}
```

Pour importer toutes les classes d'un paquetage, on utilise le mot clé « **import** » suivi du nom du paquetage et le astérisque (*). L'exemple suivant permet d'importer toutes les classes du paquetage « **java.util** » :

```
import java.util.*;
```

Remarque : Le mot-clé « **import** » doit être utilisé au début du fichier (avant la déclaration de la classe).

4) Les méthodes

Déclaration d'une méthode :

```
<------ optionnel -------> <--- obligatoire -->     <- optionnel ->
[public|protected|private] <type_retourné|void> nom([paramètres,..]) {
    // code source
}
```

Remarque : en général, une procédure est un bout de code qui réalise un traitement sans retourner un résultat. Une fonction est un bout de code qui réalise un traitement et qui retourne un résultat. En Java les procédures et les fonctions s'appellent « **méthodes** ».

Une méthode doit obligatoirement appartenir à une classe. Pour retourner une variable, la méthode utilise le mot-clé « **return** » suivi de la variable à retourner. Si la méthode ne retourne rien (**void**), on peut utiliser le mot-clé « **return** » pour quitter la méthode.

```
public void maMethode() {
    System.out.println("bonjour");
}
```

```
public void comparer(int a, int b) {
    if (a > b) {
        System.out.println(a + " supérieur à " + b);
        return;
    }
    if (a < b) {
        System.out.println(a + " inférieur à " + b);
        return;
    }
    System.out.println(a + " égale à " + b);
}
```

```
public int somme(int a, int b) {
    int resultat = a + b;
    return resultat;
}
```

```
public int max(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}
```

```
public String infoPersonne(String nom, String prenom, int age) {
    String info = "Nom:" + nom + " prénom:" + prenom + " age:" + age;
    return info;
}
```

Usage des méthodes

Pour utiliser une méthode à l'intérieur de sa classe, il suffit de l'appeler.

```
public void maMethode() {
    int s = somme(5, 6); // appel de la méthode "somme"
    System.out.println("5 + 6 = " + s);
}

public int somme(int a, int b) {
    int resultat = a + b;
    return resultat;
}
```

Pour utiliser une méthode à l'extérieur d'une classe, il faut créer une instance de la classe (un objet initialisé), et appeler la fonction à partir de l'objet.

```
Personne p = new Personne();
p.maMethode();
int s = p.somme(5, 6);
System.out.println("5 + 6 = " + s);
```

Pour utiliser une méthode « statique » à l'extérieur d'une classe, il faut l'appeler à partir du nom de la classe.

```
Personne.maMethodeStatique();
int s = Personne.sommeStatique(5, 6);
System.out.println("5 + 6 = " + s);
```

5) Les conditions « if » et « switch »

Une condition permet d'exécuter une portion de code ou non en fonction du résultat d'une ou plusieurs variables booléennes (**true** ou **false**). Pour évaluer des conditions, on utilise les opérateurs logiques.

Les opérateurs logiques :

- **==** : tester l'égalité
- **!=** : tester l'inégalité
- **<** : strictement inférieur
- **<=** : inférieur ou égal
- **>** : strictement supérieur

- **>=** : supérieur ou égal
- **&&** : l'opérateur ET
- **||** : le OU
- **!** : le NON

Exemples :

```
int a = 5, b = 6;
boolean b0 = false;
boolean b1 = a <= b;
boolean b2 = a > b;
boolean b3 = a != b;
boolean b4 = (b1 && b2) || b3;
boolean b5 = !b4;
```

Les instructions if/else :

On peut utiliser les instructions « if/else » pour vérifier des conditions

```
if(condition)
{
  //Exécution des instructions si la condition est remplie
}
else
{
  //Exécution des instructions si la condition n'est pas remplie
}
```

La condition est de type booléen (true ou false). Exemple :

```
if (a <= 5) {
    // a inférieur ou égale à 5
}
else if (a <= 10) {
    // a entre 6 et 10
}
else if (a <= 30) {
    // a entre 11 et 30
}
else {
    // a supérieur strictement à 30
}
```

L'instruction switch :

L'instruction « **switch** » peut être utilisée pour vérifier des égalités. Lorsque le nombre d'égalités est important, cette instruction permet d'alléger un code qui utilise « **if/else** ».

Syntaxe :

switch (variable)
{
  case valeur1:
  //variable = valeur1
  //action1
  break;
  case valeur2:
  //variable = valeur2
  //action2
  break;
  default:
  //action3
}
	Equivalent à:
	
	if (variable == valeur1) {
	  //action1
	}
	else if (variable == valeur2) {
	  //action2
	}
	else {
	  //action3
	}

L'instruction « **break;** » permet de sortir du « **switch** » si la condition correspond.

```
int i = 2;

switch (i) {
    case 1:
        System.out.println("Menu principal");
        afficherMenu();
        break;
    case 2:
        System.out.println("Recharger");
        afficherForfaits();
        break;
    case 3:
        System.out.println("Consulter solde");
        afficherSolde();
        break;
    case 4:
        System.out.println("Transfert argent");
        afficherOptions();
        break;
    default:
        System.out.println("Au revoir!");
}
```

Remarque : il ne faut **JAMAIS** oublier « **break** » à la fin de « **case** » sinon le programme exécute toutes les instructions (sans vérifier la/les conditions) jusqu'à trouver une instruction « **break;** » ou atteindre la fin de « **switch** ».

6) Les boucles « while » et « for »

La boucle « while »

Permet de répéter une boucle tant que la condition de répétition est vraie. Structure :

```
while (condition)
{
    instructions1;
    ...
    instructionsN;
}
```

Exemple :

```
int compteur = 5;

while (compteur > 0)
{
  //On affiche un message
  System.out.println("Bonjour " + compteur);

  compteur--;
}

System.out.println("Au revoir…");
```

Attention aux boucles infinies ! Exemple :

```
int a = 1, b = 15;
while (a < b)
{
    System.out.println("a = " + a + ", b = " + b);
}//boucle infinie
```

Pour corriger cette boucle infinie, il est possible d'ajouter une instruction permettant d'augmenter « a » (ou de réduire « b »). Exemple :

```
int a = 1, b = 15;
while (a < b)
{
    System.out.println("a = " + a + ", b = " + b);
    a++; // ou a += 1; ou a = a + 1;
}
```

La boucle for

La boucle « **for** » contient :

- Une instruction de déclaration et/ou d'initialisation d'une variable
- Une condition qui doit être remplie pour exécuter une nouvelle fois la boucle
- Une instruction qui s'exécute à la fin de la boucle

```
for (int i = 0; i < 5; i++) {
    System.out.println("ligne numéro " + i);
}
```

7) Les tableaux, les vecteurs et les listes

Les tableaux

Les tableaux en Java ont une taille fixe.

Tableaux une dimension :

```
type_du_tableau nom_du_tableau[] = {contenu_du_tableau};
type_du_tableau[] nom_du_tableau = {contenu_du_tableau};
```

Exemples :

```
int tableauEntier[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
double tableauDouble[] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0};
char tableauCaractere[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
String tableauChaine[] = {"chaine1", "chaine2", "chaine3", "chaine3"};

int tableauEntier2[] = new int[6];
```

Remarque : un tableau d'entiers doit contenir des entiers uniquement ! Et pareil pour les autres types.

Usage des tableaux :

Remarque : Un tableau de taille N débute à l'indice 0 et se termine à l'indice « N-1 »

Pour connaitre la taille d'un tableau, on utilise la variable « **length** » (une variable publique de l'objet tableau). Modification des valeurs d'un tableau :

```
int tab[] = new int[6];
for(i = 0; i < tab.length; i++) {
    tab[i] = i * 3;
}
```

```
char tab[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
tab[6] = 'l'; // l'ancienne valeur 'g' sera remplacée par 'l'
tab[7] = 'm' ; // erreur car la case 7 (8eme case) est indéfinie
```

Affichage du contenu d'un tableau :

```
char tab[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
for(int i = 0; i < tab.length; i++) { // i entre 0 et "tab.length – 1"
    System.out.println("case " + i +" contient : " + tab[i]);
}
```

Les vecteurs et les listes

Les classes Vector et List offrent une alternative pour les tableaux lorsque le nombre d'éléments est variable. En plus un vecteur ou une liste peut contenir des éléments de différents types (le même vecteur peut contenir un mélange d'entiers, de String, d'objets, etc.)

8) Les exceptions

Une exception est une erreur se produisant dans un programme qui conduit le plus souvent à l'arrêt de celui-ci. Le fait de gérer les exceptions s'appelle aussi « la capture d'exception ». Le principe consiste à repérer un morceau de code (par exemple, une division par zéro) qui pourrait générer une exception, de capturer l'exception correspondante et enfin de la traiter, c'est-à-dire d'afficher un message personnalisé et de continuer l'exécution.

Exemple de code qui génère une exception :

```
int j = 20, i = 0;
System.out.println(j/i); // division par 0
System.out.println("Bonjour");
```

Génère le message d'erreur suivant (sans exécuter la 3eme instruction) :

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
     at Exemple1.main(Exemple1.java:10)
```

La capturer de l'exception est possible avec un bloc **`try/catch`**, puis réaliser un traitement en conséquence. Par exemple, afficher un message personnalisé lors d'une division par 0.

```
int j = 20, i = 0;
try {
    System.out.println(j/i);
} catch(Exception e) {
    System.out.println("Erreur!");
}
System.out.println("Bonjour");
```

II. Java Orienté Objet

1) L'héritage

L'héritage consiste à définir une classe, dite classe dérivée ou classe enfant, à partir d'une autre classe, dite classe de base ou classe parent, en récupérant automatiquement dans la classe dérivée tous les membres de la classe de base, et en lui en ajoutant éventuellement de nouveaux membres. Par défaut toute classe hérite de la classe « **Object** ».

L'héritage est possible en Java en utilisant le mot-clé « **extends** » comme suit :

```
public class Enfant extends Parent {}
```

Exemple (classe parent) :

```
public class Personne {
    String nom;

    public Personne() {
        nom = "";
    }

    public Personne(String nom) {
        this.nom = nom;
    }

    public void afficherInformations() {
        System.out.println("Nom : " + nom);
    }
}
```

Dans cet exemple, la classe « Personne » a une variable, 2 constructeurs et une méthode.

On veut construire la classe enfant « Etudiant » qui hérite de la classe parent « Personne », tel que :

- Un Etudiant possède 2 variables : nom et etablissement. Donc on utilise la variable héritée de « Personne » et on définit une nouvelle variable etablissement.
- Etudiant possède 2 constructeurs qui initialisent les 2 variables. On utilise le mot clé « **super** » qui permet d'appeler les constructeurs du parent et d'initialiser la variable héritée. Ensuite on initialise etablissement.
- La classe « Etudiant » redéfinit la méthode « afficherInformations » de « Personne » pour afficher les informations d'un étudiant. Elle appel la méthode du parent en utilisant le mot-clé « **super** » et affiche seulement la valeur de « etablissement ».
- La classe « Etudiant » définit une nouvelle méthode « afficherDetails » qui affiche le nom et l'établissement.

Résultat (classe enfant) :

```
public class Etudiant extends Personne {
    String etablissement;

    public Etudiant() {
        //appel le constructeur du parent : Personne()
        super();
        etablissement = "";
    }

    public Etudiant(String nom, String etablissement) {
        //appel le constructeur du parent : Personne(nom)
        super(nom);
        this.etablissement = etablissement;
    }

    public void afficherInformations() {
        super.afficherInformations ();
        System.out.println("Etablissement : " + etablissement);
    }

    public void afficherDetails() {
        System.out.println("Nom : " + nom);
        System.out.println("Etablissement : " + etablissement);
    }
}
```

Exercices :

1) Créer la classe « Personne » qui contient :
    a. Les variables globales publiques suivantes :
        - String nom
        - String prenom
        - String cin
    b. Un constructeur sans paramètre qui permet d'initialiser les variables globales.
    c. Un constructeur avec des paramètres (nom, prenom, cin) permettant de renseigner toutes les variables globales.
    d. Une méthode « **afficherInformations** » permettant d'afficher les informations d'une personne.
    e. Créer la classe « **GestionnairePersonnes** » et ajouter y la méthode « main ».
    f. Créer l'objet « **personne1** » de type « **Personne** » en utilisant le constructeur sans paramètres. Appeler la méthode « **afficherInformations** » de cet objet. Quel est le message qui doit s'afficher ?

g. Créer un deuxième objet « **personne2** » de type « **Personne** » en utilisant le constructeur avec paramètres (utiliser des valeurs de votre choix). Appeler la fonction « **afficherInformations** ». Quel est le message qui doit s'afficher ?
h. Modifier le nom de « **personne2** » et appeler la fonction « **afficherInformations** ». Quel est le message qui doit s'afficher ?

2) Créer la classe « Ouvrier » qui hérite la classe « Personne » et qui a :
   a. Les variables globales privées suivantes :
      - String employeur
      - int experience
   b. Un constructeur sans paramètres qui initialise toutes les variables globales héritées et définies (appeler le constructeur sans paramètres du parent pour initialiser les variables héritées)
   c. Un constructeur avec paramètres permettant de renseigner toutes les variables globales héritées et définies (appeler le constructeur sans paramètres du parent pour initialiser les variables héritées)
   d. Redéfinir la méthode héritée « **afficherInformations** » dans la classe « Ouvrier » pour afficher toutes les informations d'un ouvrier.
   e. Créer la classe « **GestionnaireOuvriers** » et ajouter y la méthode « main ».
   f. Créer l'objet « **ouvrier1** » de type « **Ouvrier** » en utilisant le constructeur sans paramètres. Appeler la fonction « **afficherInformations** ». Quel est le message qui doit s'afficher ?
   g. Créer un deuxième objet « **ouvrier2** » de type « **Ouvrier** » en utilisant le constructeur avec paramètres (utiliser des valeurs de votre choix). Appeler la fonction « **afficherInformations** ». Quel est le message qui doit s'afficher ?

IV. Android

1) Interface graphique

Activity : une activité est une fenêtre sous Android. Elle peut contenir des vues (View). Il y a deux types de vues :

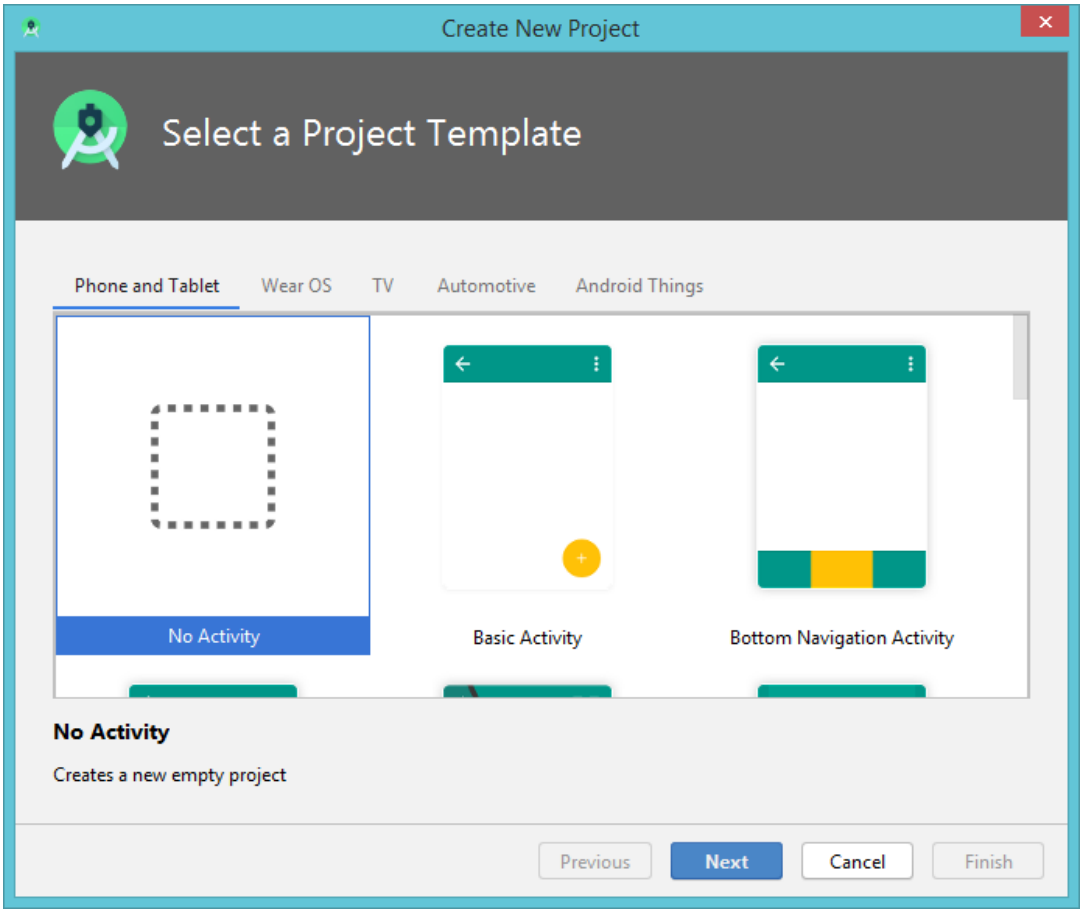
- Les vues simples, telles que :
    - Button : bouton
    - TextView : étiquette
    - EditText : champ de texte
- Les conteneurs de vues, tels que :
    - LinearLayout : un conteneur qui peut disposer les vues horizontalement ou verticalement
    - ListView : dispose les vues dans une liste

Une application peut contenir plusieurs activités. Il faut déclarer une activité par défaut qui s'affiche au lancement d'une activité.

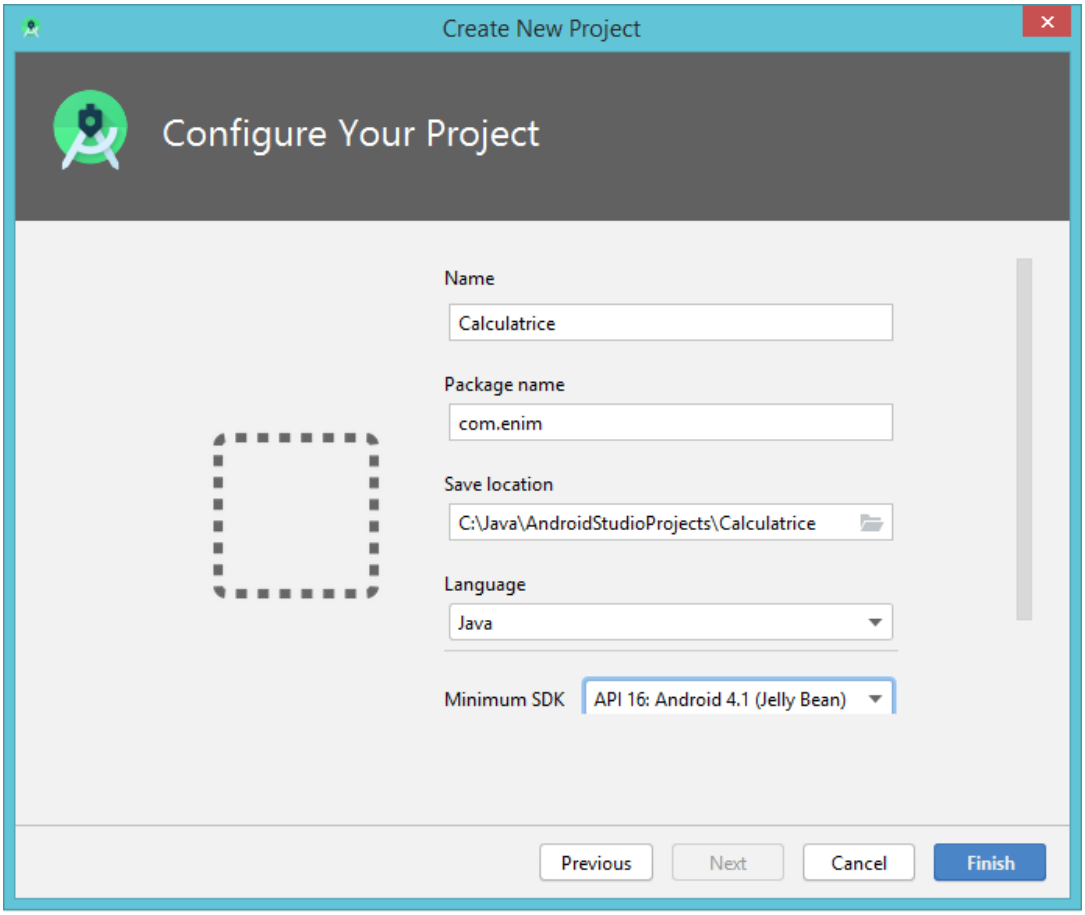
3) Création d'un projet Android

Pour créer une nouvelle application Android, il faut créer un nouveau projet sous « Android Studio » en suivant les étapes suivantes :

- Clic sur « **File > New > New Project** » et la fenêtre « **Select a Project Template** » s'ouvre :



- Sélectionner « No Activity » et appuyer sur « Next ». La fenêtre « Configure Your Project » s'ouvre



- Renseigner le nom du projet dans le champ « **Name** », le nom du package dans le champ « **Package name** », et l'emplacement de sauvegarde du projet. Choisir le langage « **Java** » et un API cible. Appuyer sur le bouton « **Finish** » pour terminer la création du projet

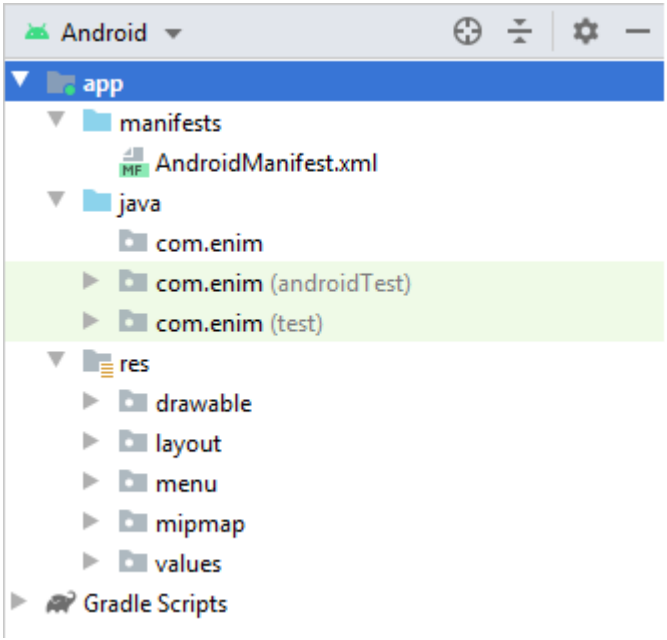
4) Structure d'un Projet Android

Chaque projet Android contient le module « **app** » qui regroupe l'ensemble des codes sources (fichiers java) et les fichiers ressources (fichiers layout, XML, images, constantes, etc.). Dans le module « app », les fichiers sont affichés dans les groupes suivants :

- **manifests** : contient le fichier « **AndroidManifest.xml** ». Ce fichier est essentiel et contient la liste des composants d'une application. Par exemple, il contient l'ensemble des activités disponibles et l'activité principale qui s'affiche au lancement de l'application.
  Remarque : si une activité n'est pas déclarée dans le fichier manifest, il n'est pas possible de l'afficher dans l'application.

- **Java** : contient tous les fichiers java

- **res** : contient tous les fichiers ressources (non java), tels que layouts, images, XML, etc.



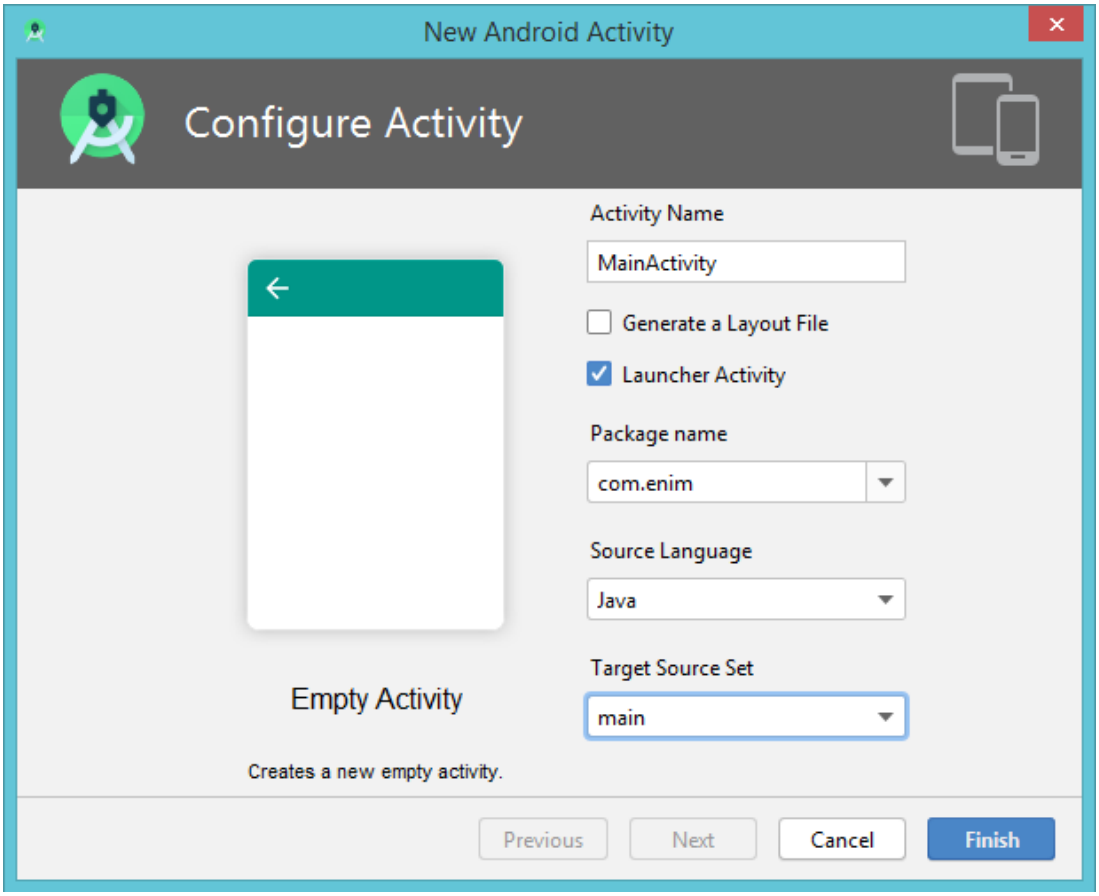
5) Ajouter des fichiers au projet

Pour ajouter un fichier au projet, vérifier d'abord que le module « app » est sélectionné.

Ajouter une activité

Pour ajouter une activité au projet, il faut suivre les étapes suivantes :

- Clic sur « **File > New > Activity > Empty Activity** », et la fenêtre « **Configure Activity** » s'affiche :



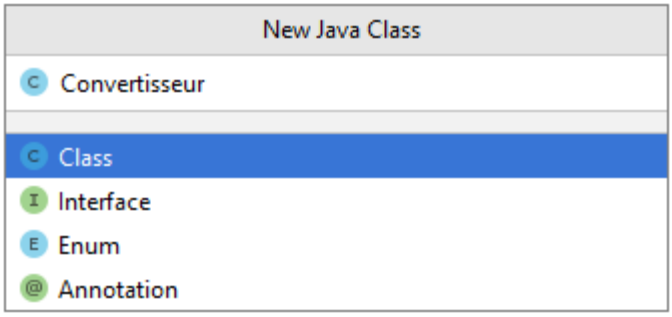
- Saisir le nom de cette activité dans le champ « **Activity Name** ». Décocher la case « **Generate Layout File** ». Si cette activité doit s'afficher au moment du lancement de l'application, cocher « **Launcher Activity** », sinon décocher cette case. Appuyer sur le bouton « **Finish** » pour terminer la création de la nouvelle activité.

Remarque : Une activité est une classe Java qui doit être déclarée dans le fichier manifest du projet. Si on ajouter cette classe en suivant les étapes précédentes, « Android Studio » se charge de la déclarer dans le fichier manifest.

Ajouter une classe Java

Pour ajouter une classe java ordinaire, il faut suivre les étapes suivantes :

- Sélectionner l'emplacement d'ajout de la classe à partir de l'arborescence du projet, exemple : « **com.enim** », et appuyer sur « **File > New > Java Class** ». La fenêtre suivante s'affiche :

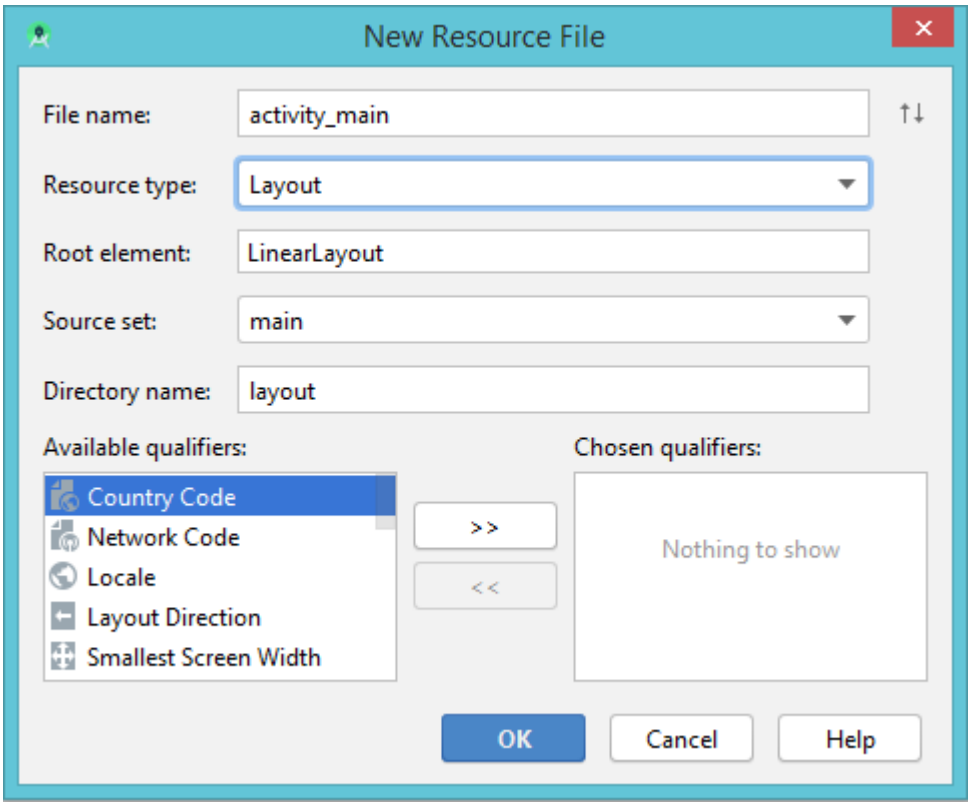


- Sélectionner le type « Class », renseigner le nom de la classe et appuyer sur « Entrer »

Ajouter un fichier Layout

Un fichier Layout permet de construire l'interface graphique en utilisant le Designer et un code XML. Pour ajouter un fichier Layout, il faut suivre les étapes suivantes :

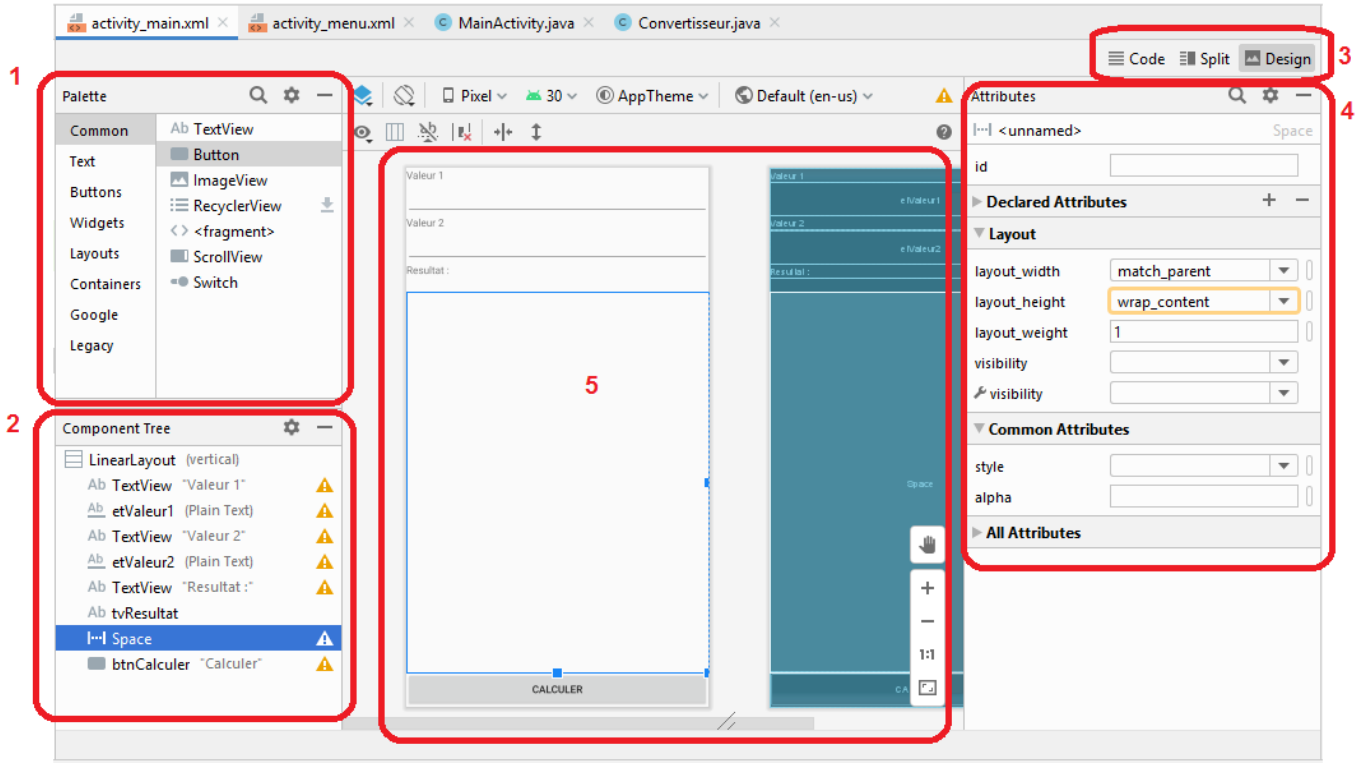
- Clic sur « **File > New > Android Resource File** », et la fenêtre suivante s'ouvre :



- Saisir le nom du fichier Layout dans le champ « **File name** » (**NE PAS** oublier l'extension XML du fichier). A partir du champ « **Resource type** », choisir le type « **Layout** ». Clic sur le bouton « **OK** ». Le nouveau fichier sera ajouter dans le groupe « **app > res > layout** ».

Construire une interface graphique

Après l'ajout d'un fichier layout, il est possible de créer l'interface graphique avec le designer à partir de la fenêtre suivante :



La **zone 1** contient les différentes vues (composants) qu'on peut ajouter à l'interface graphique.

La **zone 2** contient les différentes vues de l'interface. Il faut sélectionner une vue pour voir et modifier ses attributs à partir de **la zone 4**.

La **zone 3** permet de basculer entre le designer et le code XML du fichier Layout.

La **zone 4** permet de renseigner la valeur des différents attributs (les attributs XML) d'une vue, tel que :

- **ID** (identifiant d'une vue) : permet d'accéder à la vue à partir du code Java. Il est possible de laisser une vue sans identifiant si elle ne sera pas modifiée à partir du code Java.
- **text** : le texte qui sera affiché sur la vue
- **layout_width** : la largeur de la vue dans son conteneur parent (LinearLayout dans cet exemple)
- **layout_height** : la hauteur de la vue dans son conteneur parent. Les valeurs possibles pour **layout_width** et **layout_height** sont : **wrap_content** (taille vue = taille de son contenue) et **match_parent** (taille vue = taille restant du parent : impossible d'ajouter d'autres vues).
- **layout_weight** (la part d'une vue de son conteneur parent) : si une seule vue a un **layout_weight** dans un LinearLayout, elle occupe tout l'espace libre du LinearLayout. Sinon, l'espace libre sera partagé selon la part de chaque vue (exemple : **layout_weight=1** pour vue1 et **layout_weight=2** pour vue2 implique vue1 occupe 33% et vue2 occupe 66% de l'espace libre).

Par défaut, la **zone 4** affiche les attributs principaux d'une vue. Pour afficher tous les attributs, il faut appuyer sur « **All Attributes** ».

Le designer permet de compléter le code XML correspondant à l'interface. Pour consulter le code XML, appuyer sur « Code » à partir de la **zone 3**. Exemple de code :

```
<Button
    android:id="@+id/btnAfficher"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Afficher" />
```

La balise XML « **Button** » contient les différents attributs de la vue « **Button** ».

6) Afficher un fichier Layout dans une activité (Activity)

Pour afficher une interface graphique (GUI : Graphical User Interface) dans une activité, on appel la fonction suivante:

```
setContentView(R.layout.nom_fichier_layout_sans_extension);
```

L'accès à partir d'une activité aux différents composants d'un layout se fait moyennant leurs identifiants. Cela est possible comme suit :

```
findViewById(R.id.idVue); // idVue est l'identifiant renseigné dans
                          // l'attribut XML « android:id »
```

Exemple :

```
EditText etNom = findViewById(R.id.etNom);
EditText etPrenom = findViewById(R.id.etPrenom);
TextView tvMessage = findViewById(R.id.tvMessage);
```

7) Ajouter un listener à un bouton

Pour ajouter un listener à un bouton, il est possible d'utiliser la fonction « **setOnClickListener(OnClickListener l)** » d'un bouton comme suit :

```
    btnAfficher.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // code
        }
    });
```

Exercice – Partie 1

8) Lancement d'une nouvelle activité

Il est possible de lancer une nouvelle activité (nommé activité enfant) à partir d'une activité parent. Pour quitter une activité, on peut utiliser la fonction « **finish()** ».

Sans attendre un résultat

Pour lancer une nouvelle activité sans attendre un résultat, on utilise la classe « **Intent** » et on lui donne le nom de l'activité à lancer. Ensuite on appelle la fonction « **startActivity** ».

Exemple de lancement de l'activité « **AjouterCommandeActivity** » :

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

Il est possible d'envoyer des données à l'activité qui sera affichée (activité enfant) à travers la méthode « **putExtra** » de l'intent de lancement. Cette méthode reçoit des pairs <**clé, valeur**>. Exemple :

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

intent.putExtra("description", "Nouvelle commande pour Tunis");
intent.putExtra("client", "Ahmed Abid");
intent.putExtra("numero", 1142512);
intent.putExtra("paiement", false);

startActivity(intent);
```

Les données envoyées par l'activité parent peuvent être récupérées par l'activité enfant à partir de la méthode « **onCreate** ». Exemple :

```
Intent intent = getIntent();
String strDesc = intent.getStringExtra("description");
String strClient = intent.getStringExtra("client");
int num = intent.getIntExtra("numero", 0);
boolean payer = intent.getBooleanExtra("paiement", false);
```

La méthode « **intent.getIntExtra** » prend en paramètre, la clé et une valeur par défaut. La valeur par défaut sera retournée si la clé est introuvable. Mais si la clé existe, c'est la valeur correspondante à cette clé qui sera retournée. Même remarque pour les autres méthodes de l'intent qui demandent une clé et une valeur par défaut.

Exercice – Partie 2

Avec attente de résultat

Si l'activité parent a besoin d'un résultat de l'activité enfant, il est nécessaire de :

- Côté activité parent :
  - Lancer l'activité enfant avec la méthode « **startActivityForResult** » (il faut toujours utiliser un intent)
  - Reimplémenter la méthode « **onActivityResult** » de l'activité parent qui sera exécutée à la fermeture de l'activité enfant (cette méthode reçoit le résultat de l'activité enfant)
- Côté activité enfant :
  - L'activité enfant doit sauvegarder les données dans un intent et le retourner au parent avec la méthode « **setResult** ». Avec les données retournées, il faut retourner le résultat (RESULT_OK ou RESULT_CANCELED).

Exemple :

Côté activité parent :

- Lancement de l'activité enfant :

```
Intent intent = new Intent(this, AjouterCommandeActivity.class);
startActivityForResult(intent, 0);
```

- Reimplémentation de la méthode « **onActivityResult** » :

```
protected void onActivityResult(int req, int result, Intent data)
{
    if (result == RESULT_OK) {
        //code au cas de succés
        //lire données de Intent "data" et faire le traitement
    }
    else {
        //code si annulé
    }
}
```

Côté activité enfant :

- Si le résultat est OK (exemple : appui sur le bouton OK de l'activité)

```
void operationOk() {
    Intent intent = new Intent(); // pour sauvegarder les données
    intent.putExtra("operation", 2);
    setResult(RESULT_OK, intent);
    finish();
}
```

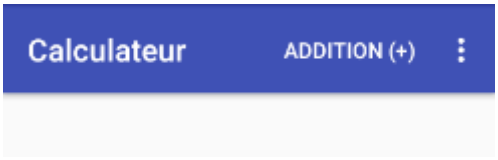
- Si l'opération est annulée (exemple : appui sur le bouton Annuler de l'activité)

```
void operationAnnuler() {
    Intent intent = new Intent();
    setResult(RESULT_CANCELED, intent);
    finish();
}
```

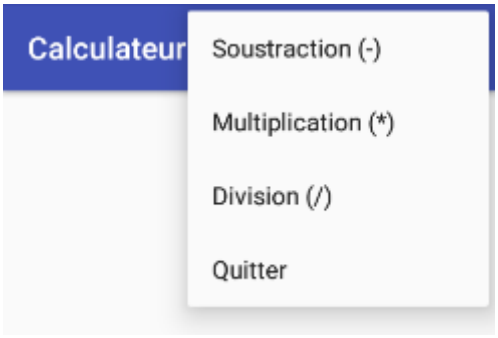
Exercice – Partie 3

9) Création d'un menu

Un menu s'affiche dans la barre de titre (ActionBar) de l'activité. Un menu contient des « items ». Un item peut s'afficher sur la barre de titre ou peut être masqué. Pour afficher les items masqués, on appui sur les « 3 points verticaux » du menu. Exemple de menu :



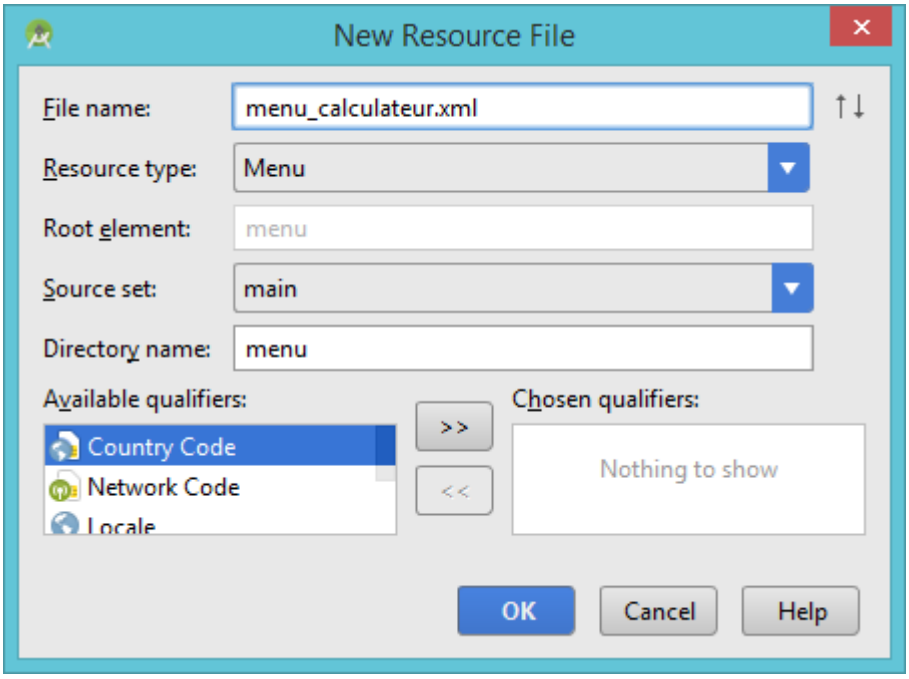
L'item « ADDITION (+) » est affiché sur la barre de titre, et les autres items du menu sont accessibles à travers les « 3 points verticaux »



Les items masqués du menu

Il est possible de construire le menu d'une activité en utilisant un fichier ressource de type « Menu » et le Designer. Pour ajouter un fichier Menu, il faut suivre les étapes suivantes :

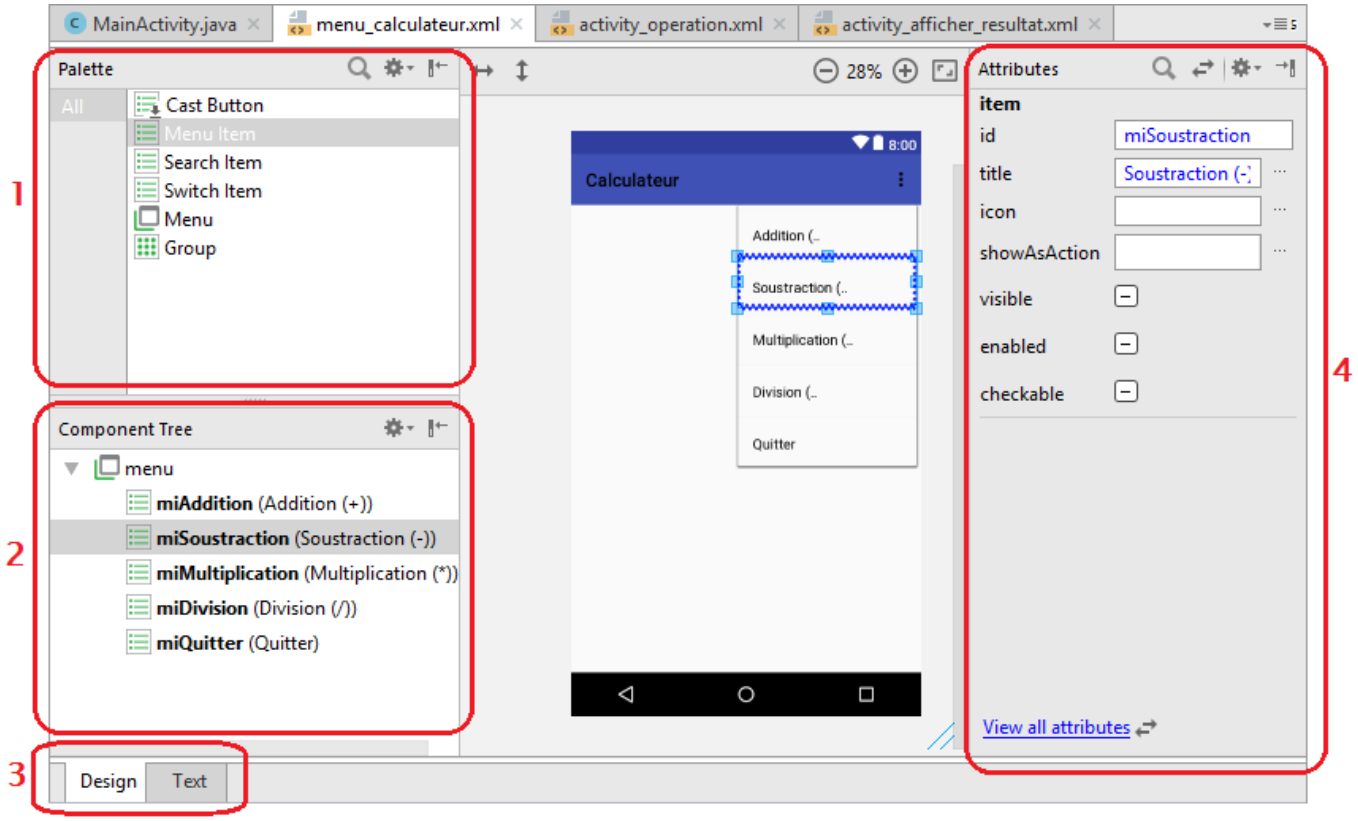
- Clic sur « **File > New > Android Resource File** », et la fenêtre suivante s'ouvre :



- Saisir le nom du fichier Menu dans le champ « **File name** » (**NE PAS** oublier l'extension XML du fichier). A partir du champ « **Resource type** », choisir le type « **Menu** ». Clic sur le bouton « **OK** ». Le nouveau fichier sera ajouter dans le groupe « **app > res > menu** ».

Construire un Menu

Après l'ajout d'un fichier Menu, il est possible de créer un menu avec le designer à partir de la fenêtre suivante :



La **zone 1** contient les différents composants disponibles. Un « **Menu Item** » est un item qu'on peut ajouter à un menu.

La **zone 4** permet de renseigner la valeur des différents attributs d'un item, tel que :

- **ID** (identifiant) : permet d'accéder à un item à partir du code Java
- **title** : le texte de l'item
- **showAsAction** : indique comment l'item sera affiché dans la barre de titre. Exemple de valeurs possibles sont : never (l'item sera toujours masqué), ifRoom (afficher l'item sur la barre de titre s'il y a de l'espace), always (l'item sera toujours affiché dans le barre de titre).

Afficher le menu dans l'activité

Pour afficher le menu du fichier ressource dans l'activité, il faut redéfinir la méthode héritée « **onCreateOptionsMenu** ». Cette méthode est appelée par le système au moment de la création du menu de l'activité.

Exemple :

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater mi = getMenuInflater();
    mi.inflate(R.menu.menu_calculateur, menu);

    return true;
}
```

Ajouter un listener aux items :

Lors de l'appui sur un item, le system appelle la méthode « **onOptionsItemSelected** » et lui fournit l'item sélectionné. Il faut redéfinir cette méthode pour gérer les évènements lors de l'appui sur les items.

Exemple :

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.miAddition:
            Toast.makeText(this, "item +", Toast.LENGTH_LONG).show();
            this.operation = 1;
            break;
        case R.id.miSoustraction:
            Toast.makeText(this, "item -", Toast.LENGTH_LONG).show();
            this.operation = 2;
            break;
        case R.id.miQuitter:
            finish();
            break;
    }

    return true;
}
```

Exercice – Partie 4

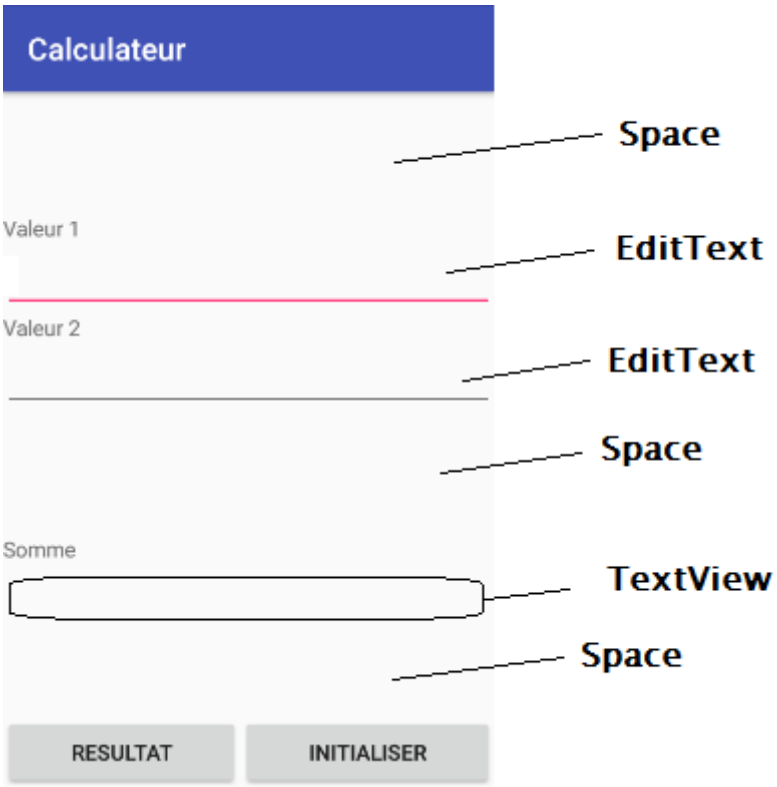
Exercice :

Partie 1

a) Créer un nouveau projet Android nommé « **Calculateur** » dans le package « **com.enim** ». A partir de la fenêtre « Select a Project Template », choisir « No Activity ».

b) Ajouter une nouvelle activité nommée « **CalculateurActivity** » au projet, avec :
   - Generate Layout File : décoché
   - Launcher Activity : coché

c) Ajouter un nouveau fichier Layout nommé « **activity_calculateur.xml** » au projet, permettant d'afficher l'interface suivante :



d) Afficher ce fichier layout dans l'activité créée avec la fonction « **setContentView** »

e) Ajouter un listener au bouton « **RESULTAT** » permettant d'afficher la somme de « **valeur 1** » et « **valeur 2** » dans le label « Résultat ». Pour obtenir le texte de l'EditText « editText », on peut appeler l'instruction : `String txt = editText.getText().toString();`
Utiliser la méthode « `Double.parseDouble(txt)` » pour convertir une chaine en un double.

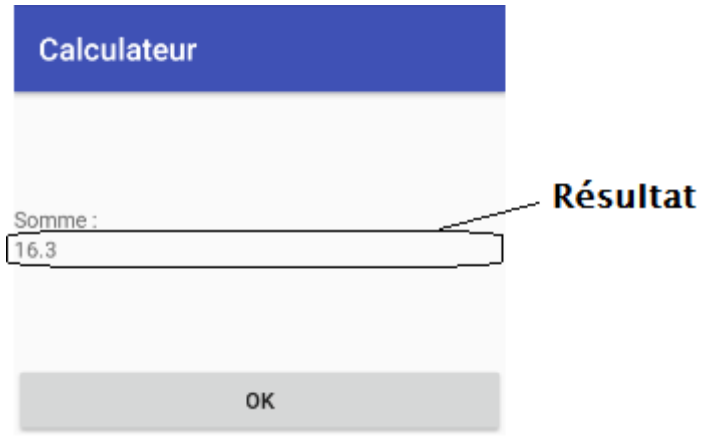
f) Afficher un Toast affichant le message « ok » lors de l'appui sur le bouton « **RESULTAT** ».
Exemple de Toast :
`Toast.makeText(this, "Bonjour", Toast.LENGTH_SHORT).show();`

g) Ajouter un listener au bouton « **INITIALISER** » permettant de vider le contenu des champs de texte « **valeur 1** » et « **valeur 2** ».

Partie 2 :

h) Ajouter une nouvelle activité nommée « **AfficherResultatActivity** », avec :
   - Generate Layout File : **décoché**
   - Launcher Activity : **décoché**

i) Ajouter un nouveau fichier Layout nommé « **activity_afficher_resultat.xml** » au projet, permettant d'afficher l'interface suivante :



j) Afficher ce fichier layout dans l'activité créée en utilisant la méthode « **setContentView** »

k) L'activité « **AfficherResultatActivity** » doit s'afficher lorsque l'utilisateur appui sur le bouton « **RESULTAT** » de « **CalculateurActivity** », et doit afficher le résultat de calcule.

l) Ajouter un listener au bouton « **OK** » de « **AfficherResultatActivity** » permettant de quitter l'activité (appelle la méthode « **finish()** »).

Partie 3 :

m) Ajouter une nouvelle activité nommée « **OperationActivity** », avec :
   - Generate Layout File : **décoché**
   - Launcher Activity : **décoché**

n) Ajouter un nouveau fichier Layout nommé « **activity_operation.xml** » au projet, permettant d'afficher l'interface suivante :

o) Afficher ce fichier layout dans l'activité créée en utilisant la méthode « **setContentView** »
p) Ajouter le bouton « **Operation** » en bas de « **activity_calculateur.xml** », et ajouter un listener à ce bouton permettant d'afficher l'activité « **OperationActivity** » (en utilisant la méthode « **startActivityForResult** »)
q) Ajouter un listener au bouton « **+** » de « **OperationActivity** » permettant de quitter l'activité et de retourner « **RESULT_OK** » et « operation = 1 » :
```
intent.putExtra("operation", 1);
```
r) Ajouter un listener au bouton « **-** » permettant de quitter l'activité et de retourner « **RESULT_OK** » et « operation = 2 »
s) Ajouter un listener au bouton « ***** » permettant de quitter l'activité et de retourner « **RESULT_OK** » et « operation = 3 »
t) Ajouter un listener au bouton « **/** » permettant de quitter l'activité et de retourner « **RESULT_OK** » et « operation = 4 »
u) Ajouter un listener au bouton « **ANNULER** » permettant de quitter l'activité et de retourner « **RESULT_ CANCELED** »
v) Implémenter la méthode « **onActivityResult()** » dans l'activité parent (**CalculateurActivity**) comme suit : **si** « **RESULT_OK** » **alors** afficher Toast avec le message « **opération = ?** », **sinon** afficher Toast avec le message « **Annuler** ». Tester.
w) Modifier la méthode « **onActivityResult()** » dans l'activité parent (**CalculateurActivity**) à nouveau afin de mémoriser le type de l'opération et de l'appliquer pour les nouvelles opérations. Tester.

Partie 4 :

x) Ajouter un nouveau fichier Menu nommé « **menu_calculateur.xml** » permettant d'afficher les items suivants :

y) Redéfinir la méthode « **onCreateOptionsMenu** » de « **CalculateurActivity** » afin d'ajouter le menu du fichier « **menu_calculateur.xml** » dans l'activité.

z) Redéfinir la méthode « **onOptionsItemSelected** » de « **CalculateurActivity** » afin de détecter les appuis sur les différents items du menu et de réaliser les taches suivantes :

- Clic sur « **Addition (+)** » : afficher un Toast avec le message « **Opérateur + sélectionné** », sélectionner l'opérateur « + » et calculer la **somme** lors de l'appui sur le bouton « **RESULTAT** ».
- Clic sur « **Soustraction (-)** » : afficher un Toast avec le message « **Opérateur - sélectionné** », sélectionner l'opérateur « - » et calculer la **différence** lors de l'appui sur le bouton « **RESULTAT** ».
- Clic sur « **Multiplication (*)** » : afficher un Toast avec le message « **Opérateur * sélectionné** », sélectionner l'opérateur « * » et calculer le **produit** lors de l'appui sur le bouton « **RESULTAT** ».
- Clic sur « **Division (/)** » : afficher un Toast avec le message « **Opérateur / sélectionné** », sélectionner l'opérateur « / » et calculer la **division** lors de l'appui sur le bouton « **RESULTAT** ».
- Clic sur « **Quitter** » : quitter l'application en appelant la méthode « **finish()** ».