Objectifs du TP :

- S'approprier l'EDI Android Studio
- Créer une première activité

Contexte

La société Vanille commercialise des confiseries. Il y a quelques mois, elle a demandé à la société StartInfo le développement et la mise en place d'un site internet spécifique pour la vente de ses produits. Le coût assez modeste de la prestation a pu être assez facilement amorti par un regain des ventes.

Aujourd'hui, la société Vanille envisage une nouvelle évolution après un retour des clients signifiant la difficulté de passer des commandes via leur téléphone.
Mise à nouveau à contribution, StartInfo a proposé un nouveau processus qui a été validé par la société Vanille:
Elaborer une application embarquée sur une solution technique d'accès (STA) sous Android, permettant de passer les commandes de bonbons.

Les principales fonctionnalités sont :

- Création d'une commande
- Import du catalogue des bonbons depuis le serveur web,
- Affichage des bonbons et sélection,
- Saisie de la quantité et ajout au panier,
- Saisie des informations du client,
- Export des données sur le serveur web.

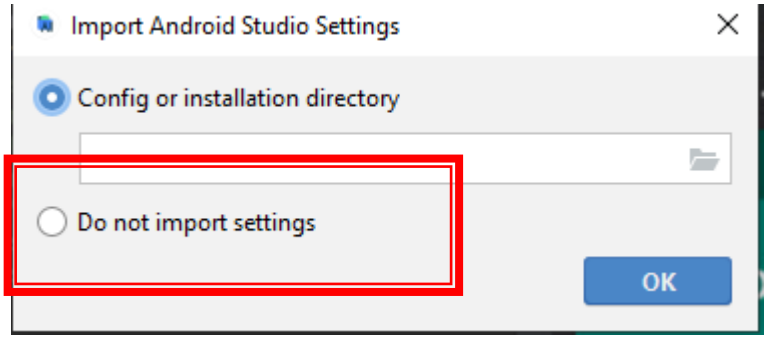
Voici le premier écran que nous allons élaborer lors de ce TP :
Visualisation d'un message de bienvenue
Et création de la commande

I. Mise en place de l'environnement de travail

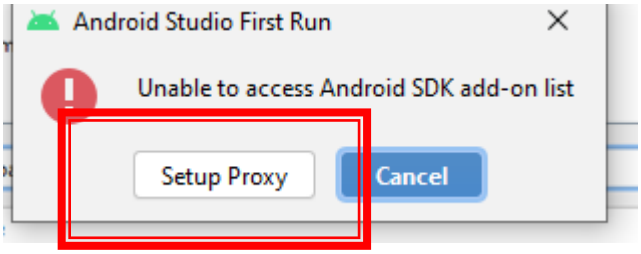
Toujours, lancer Android Studio en tant qu'administrateur

A. Premier lancement d'Android Studio afin de finaliser l'installation

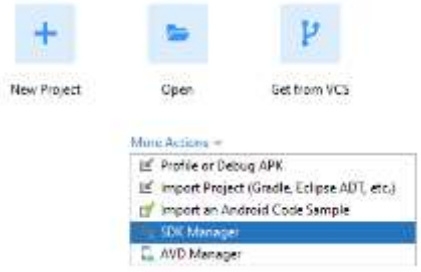
- Lancer Android Studio.
- Sélectionner « Do not import settings ».
- Choisir Do not import settings



- Lorsque le programme vous le demande, enregistrer les paramètres du proxy.



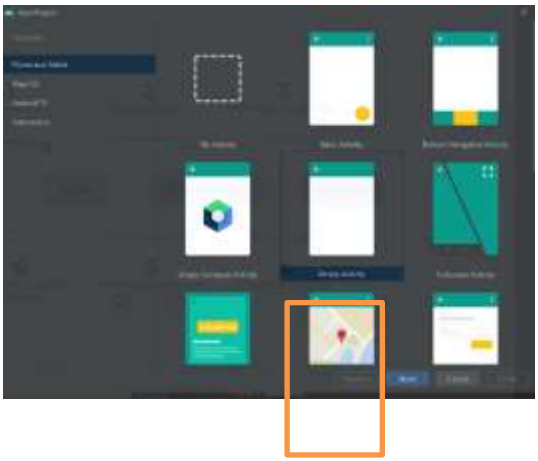
- Terminer l'installation
- Cliquer sur SDK manager afin de vérifier qu'il soit bien installé.



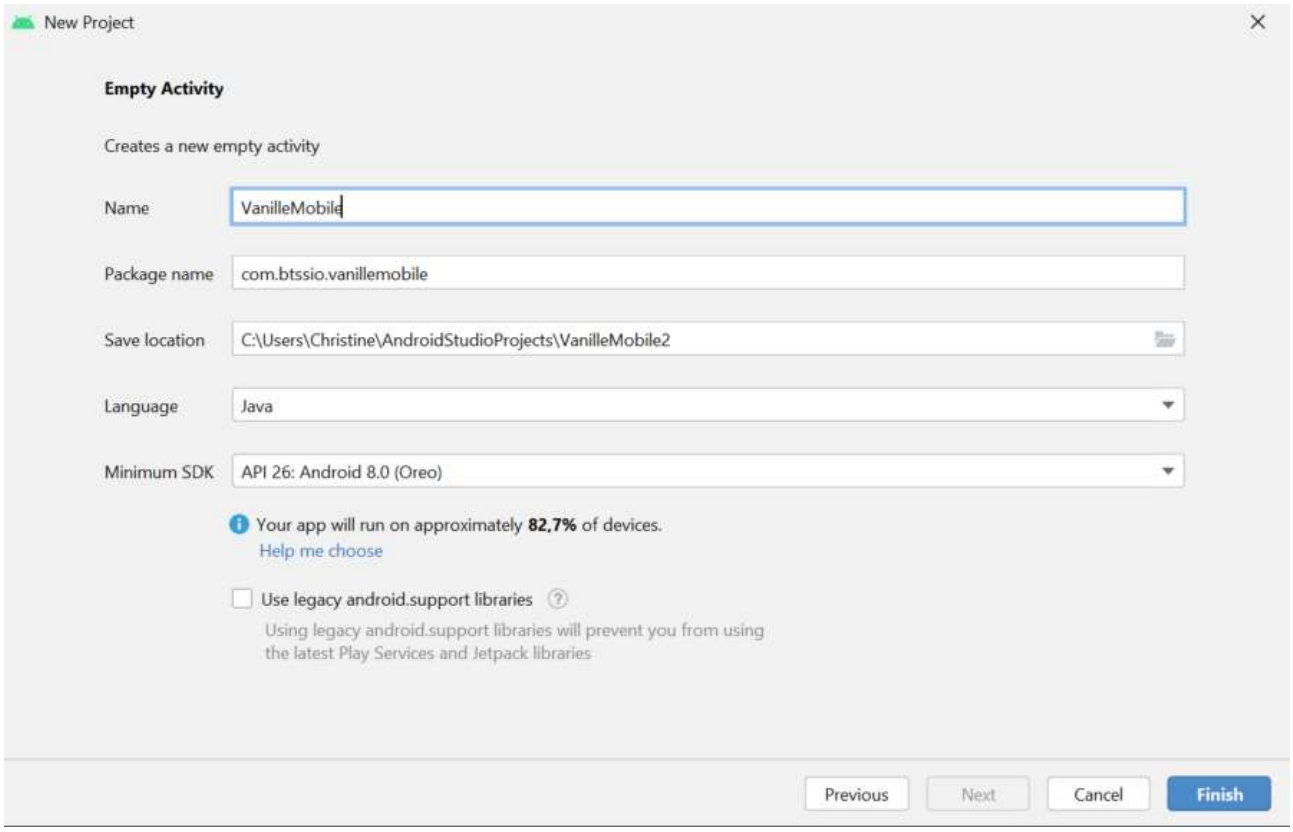
B. Création d'un projet

- Créer un nouveau projet

- Si l'IDE vous signale des mises à jour cliquer sur update (choisir update and restart)
  - Démarrer un nouveau projet nommé Vanille
  - Choisir une activité vide



- Choisir la configuration du projet comme suit :

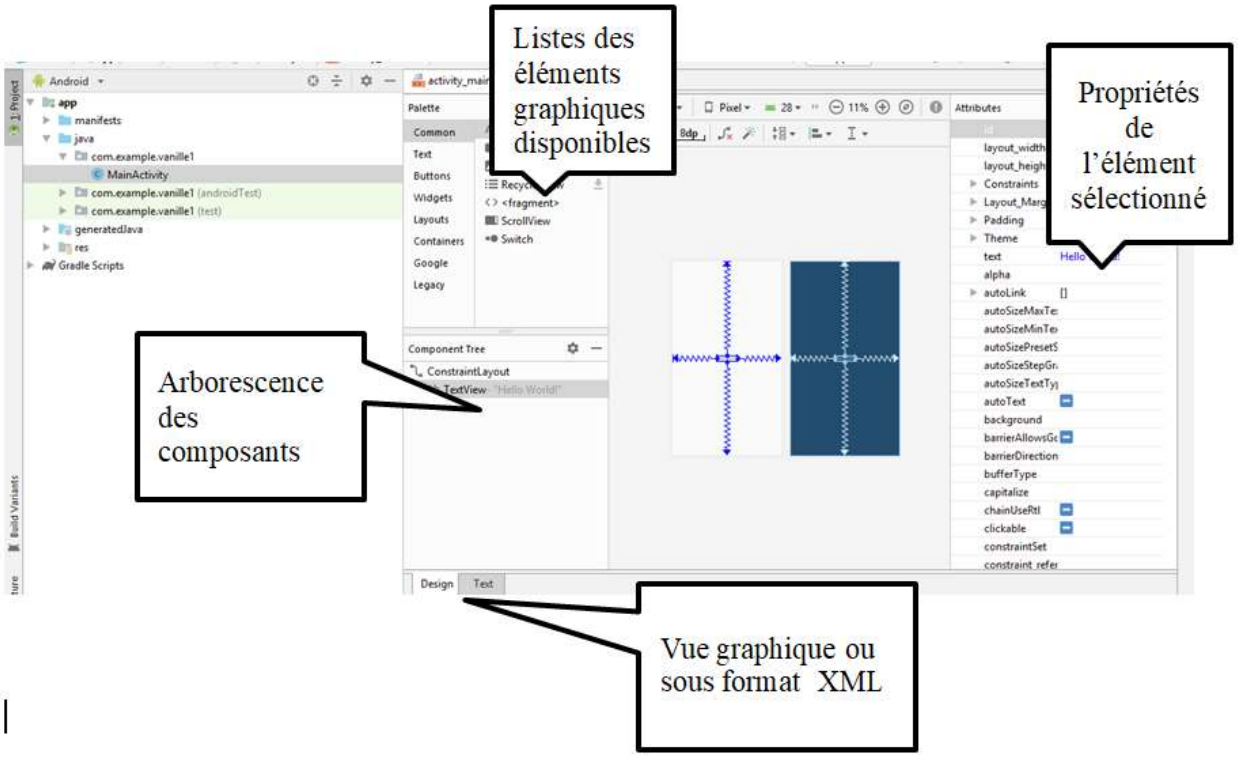


Lors de la création d'un projet se créé automatiquement une nouvelle activité composée en 2 fichiers : un fichier .XML (vue) et un fichier .java (controller)

Le fichier MainActivity.java permet d'activer son Layout (écran ou vue structurée sous Android)
Le fichier activity_main.xml permet de définir l'interface de l'activité

- Cliquer sur activity_main.xml

L'IDE propose un environnement de développement par défaut composé de plusieurs fenêtres.
Lorsque le fichier .XML est sélectionné vous avez accès à celui-ci:

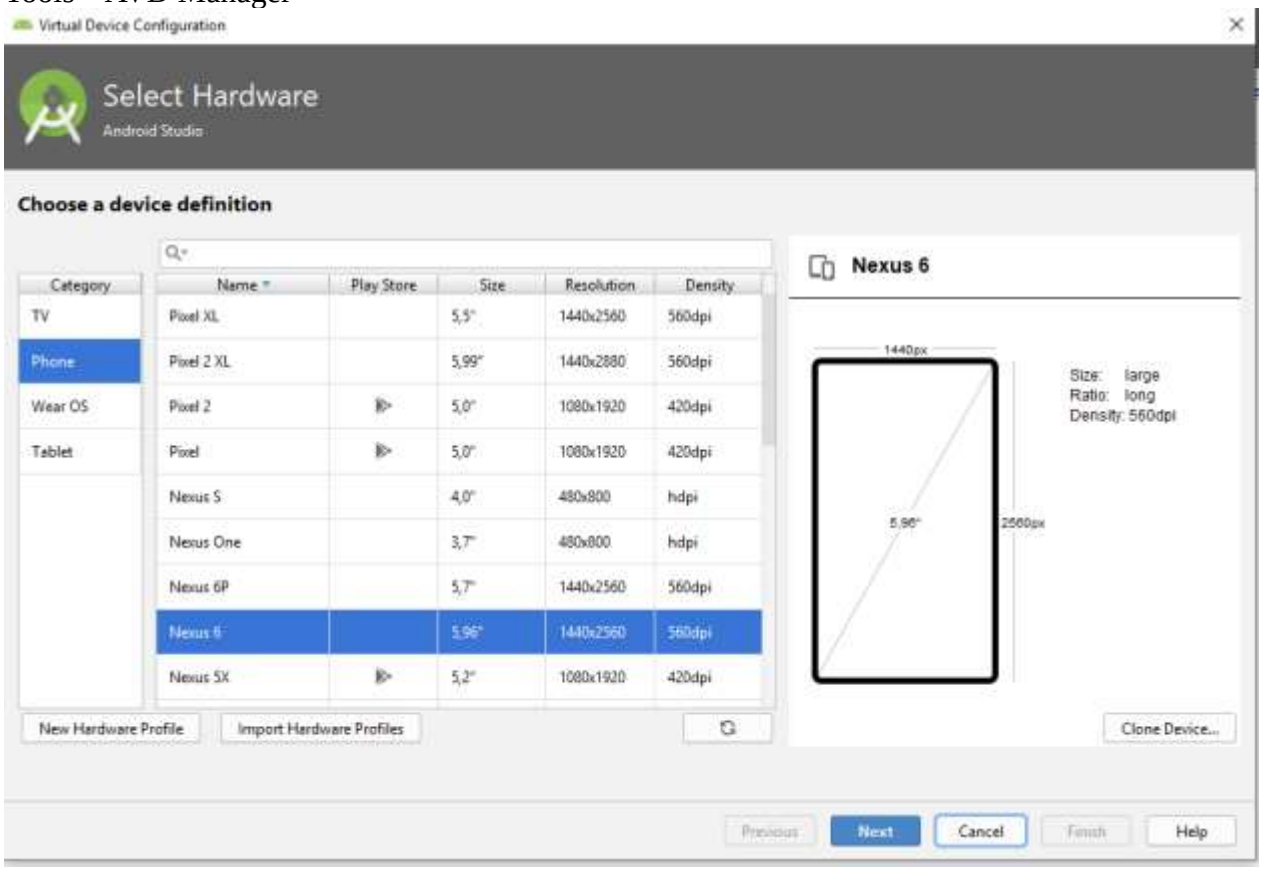


Afin de pouvoir lancer notre première application nous devons créer un AVD (émulateur)

C. Création d'un AVD (émulateur)

- Ajouter un nouveau ADV

Tools – AVD Manager



- Choisir la version du système (API 26 au minimum), la télécharger si nécessaire.
- Donner un nom à votre émulateur (significatif de préférence) .
- Lancer votre émulateur (penser à laisser cette « machine » allumée tout le temps).
- Lancer l'application. Run – Run monApplication, choisir l'émulateur et attendre….

Ne jamais fermer l'émulateur !!!!

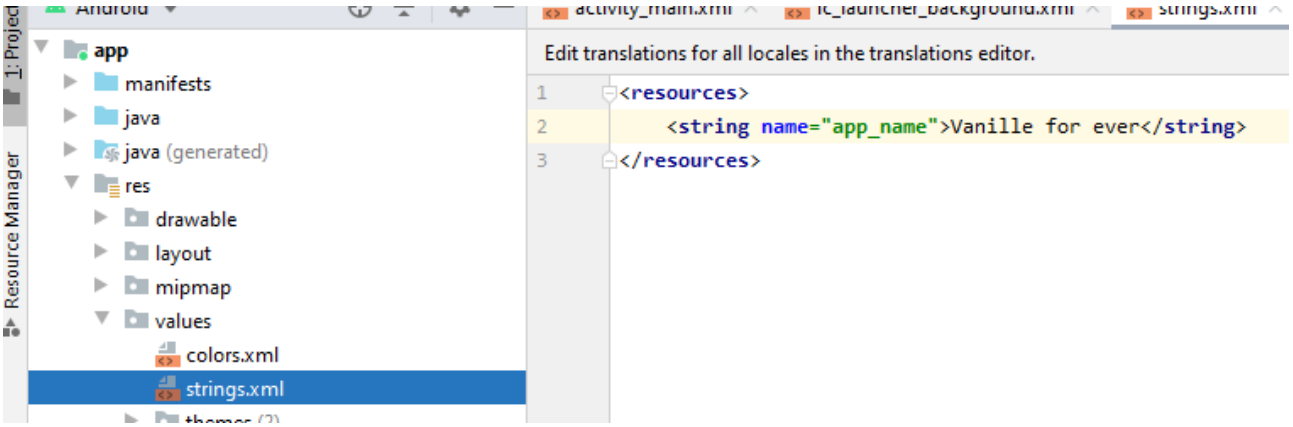
D. Mise en place de l'affichage

Nous allons modifier le message du textView.

- Modifier la Propriété text du TextView« « Bienvenue dans Vanille».
- Relancer l'application pour voir la modification.

Nous allons modifier le nom de l'application. Celui-ci est une ressource.

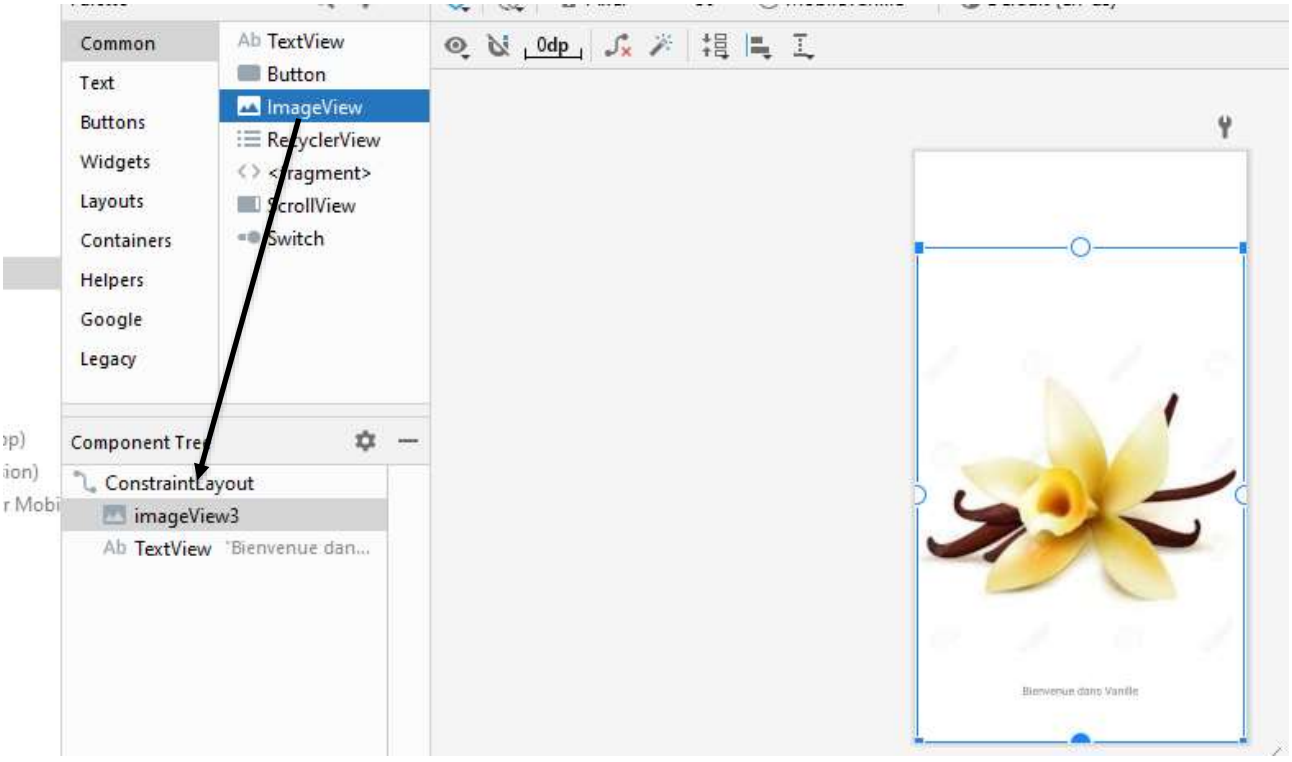
- Ouvrir le fichier strings.xml dans res/values/



- Modifier le nom de l'application.
- Relancer l'application.

Nous allons ajouter l'image représentant la fleur de vanille.

- Ajouter le fichier fleur_vanille.jpg dans le répertoire res/drawable de votre projet
- Ajouter une imageView à votre écran (faire glisser dans votre arbre)



- Relancer votre application.

II. Elaboration de notre projet

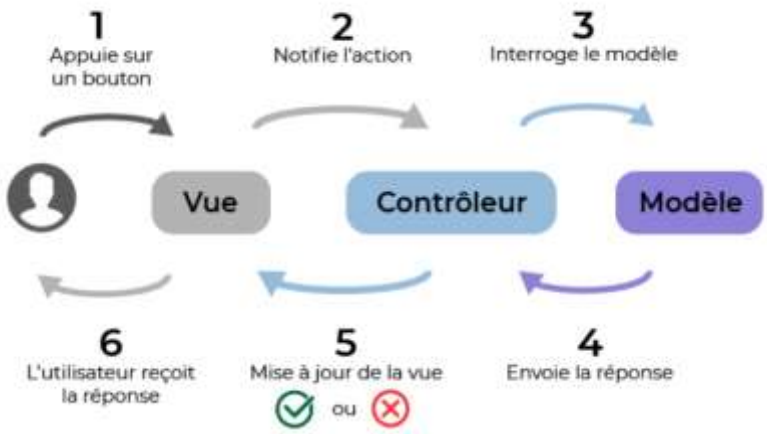
Les développeurs débutant Android (mais pas qu'eux !) utilisent l'architecture appelée MVC, pour **Model-View-Controller**, (ou *Modèle-Vue-Contrôleur*, en français). Cette architecture, bien que vieillissante, est encore très utilisée.
Pour les petits curieux, l'architecture recommandée par Google pour Android est le MVVM (pour Model-View-ViewModel). Pour le moment, concentrons-nous sur le MVC, plus simple à aborder !
Rappel de l'architecture MVC

L'architecture MVC consiste à découper son code pour qu'il appartienne à l'une des trois composantes du MVC. Lorsque vous créez une nouvelle classe ou un nouveau fichier, vous devez donc savoir à quelle composante il appartient :

Modèle : contient les données de l'application et la logique métier. Par exemple, les comptes des utilisateurs, les produits que vous vendez, un ensemble de photos, etc. La composante *modèle* n'a aucune connaissance de l'interface graphique.

Vue : contient tout ce qui est visible à l'écran et qui propose une interaction avec l'utilisateur. Par exemple, les boutons, les images, les zones de saisie, etc. Exemple : le fichier *activity_main.xml*

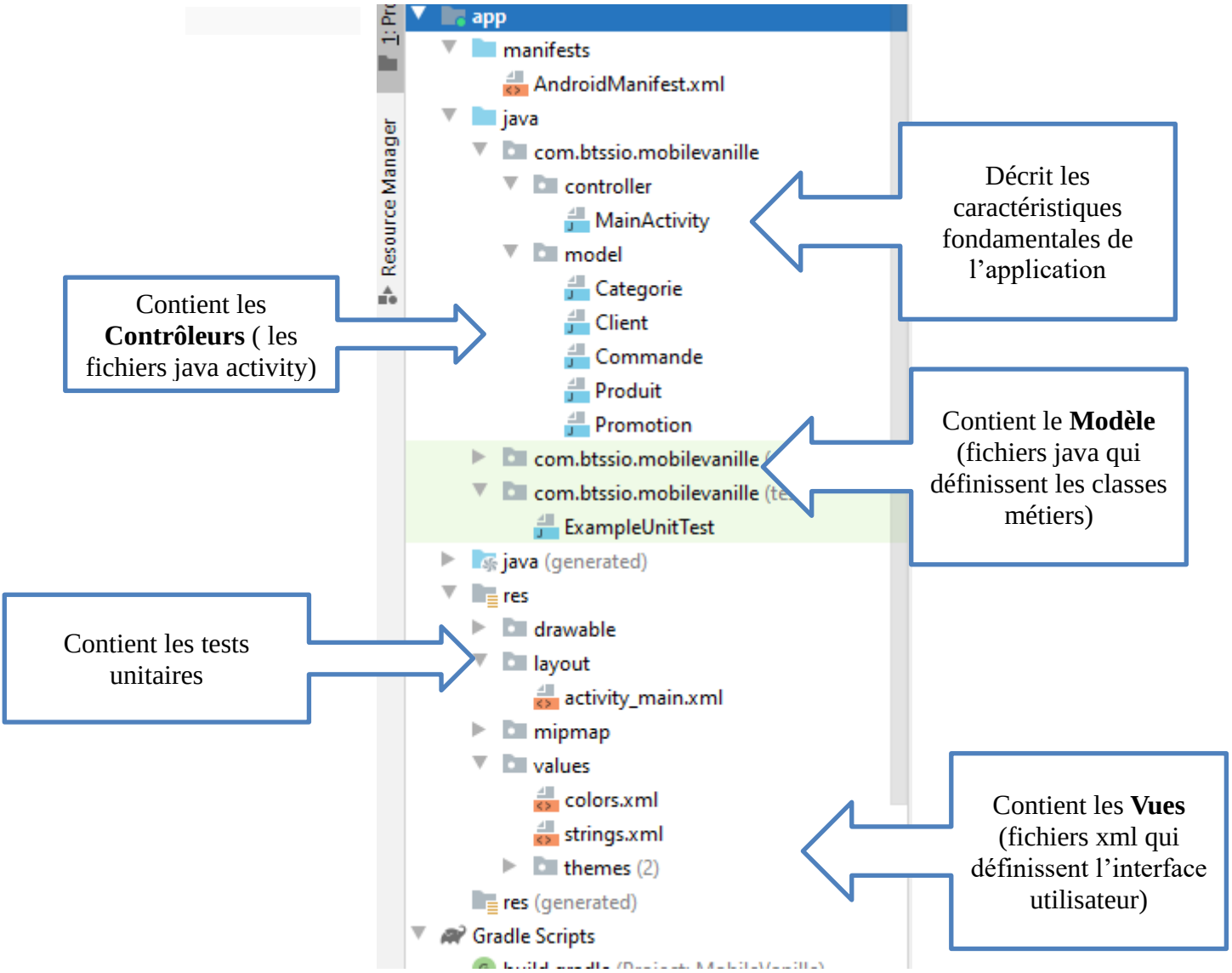
Contrôleur : c'est la "colle" entre la vue et le modèle, qui gère également la logique de l'application. Le contrôleur permet de réagir aux interactions de l'utilisateur et de lui présenter les données qu'il demande. Et ces données, où les récupère-t-il ? Dans le modèle, bien entendu !
Exemple : *MainActivity.java*



Les différentes phases d'application du modèle MVC lorsque l'utilisateur intéragit avec l'application

Au cours de l'application nous allons respecter le modèle MVC. Pour cela nous allons créer 2 nouveaux packages (model et controller) dans notre package de travail.

- Au niveau du package com.btssio.vanille, faire un clic droit, et sélectionner new – package
- Faire glisser MainActivity dans le package controller (traitement de l'interface) et Factoriser

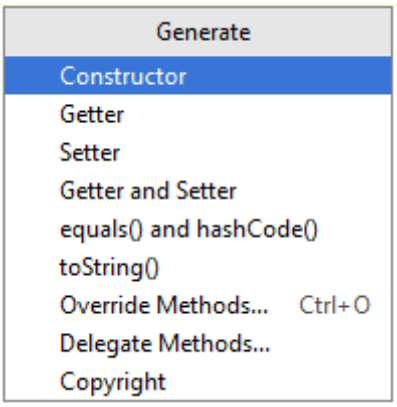


A. Elaboration du Modèle (les classes métier)

Quelques astuces de l'IDE et règles de programmation en java

Le Nom des classes commence par une majuscule,
Tout le reste commence par une minuscule (privé ou public)

Pour générer du code automatiquement, faire un clic droit dans la classe, sélectionner generate dans le menu conceptuel puis choisir les actions :



Notre modèle va comprendre 6 classes : Produit, Promotion, Catégorie, Commande, Client et Catalogue.

1. Les classes déjà codées

- Dans le package model copier les classes Produit, Promotion, Catégorie et Catalogue qui sont fournies dans les ressources directement dans le dossier.

Afin de vous remémorer le code Java et les classes proposées. Vous allez documenter ces 4 classes.
Pour permettre la génération de la documentation technique :
Au-dessus de la signature de la méthode saisir /** puis valider : génération automatique de la structure à remplir :

```
/**
 *
 * @param id
 * @param description
 * @param image
 * @param prix
 * @param categorieDuProduit
 */
public Produit(String id, String description, String image, float prix, Categorie categorieDuProduit) {
    super();
    this.id = id;
    this.description = description;
    this.prix = prix;
    this.image = image;
    this.categorieDuProduit = categorieDuProduit;
    this.lesPromos = new ArrayList<Promotion>();
}
```

- Documenter les classes et les méthodes.

2. La classe Client

- Ajouter au Modèle, la classe Client disponible dans les ressources.
- Générer les accesseurs.
- Générer le constructeur.
- Générer la méthode toString() .

3. La classe Commande

- Ajouter au Modèle la classe commande disponible dans les ressources.

L'attribut dateCommande est accessible en lecture uniquement, tous les autres sont accessibles en lecture et écriture.

- Générer les accesseurs.
- Ajouter la méthode getTotalCommande() et commenter :

```
public float getTotalCommande(){
    float total = (float) 0.0;
    for(Map.Entry<Produit,Integer> entry : lignesCommande.entrySet()) {
        total += (float) (entry.getValue() * (entry.getKey().getPrixActuel(LocalDate.now())));
    }
    return total;
}
```

Notre application a pour but de créer une commande de produits. Il faut s'assurer qu'une seule commande est créée afin de ne pas perdre des informations.

B. Elaboration d'un Singleton

Nous avons besoin de restreindre l'instanciation de la classe Commande à un seul objet dans toute l'application (notion de singleton).

Un singleton est un Design pattern très utilisé (un peu trop parfois)
Pour ce faire 3 éléments importants :

- Un attribut **static** de type de la Classe qui crée l'objet unique de la classe.
- Un getter qui retourne l'objet.
- Un constructeur de la classe en **privé** (interdit l'instanciation d'un objet à l'extérieur de classe)

Nous allons modifier la classe Commande afin de rendre cela possible.

- Faire les 2 modifications suivantes :

```
//la classe doit etre final afin d'empecher l'héritage de la classe qui pourrait permettre de contourner le singleton
public final class Commande {

    // création d'un attribut statique permettant l'accés à la classe commande
    private static Commande instance = null;
    private int id;
    private Long dateCommande;
    private Client leClient;
    private HashMap<Produit, Integer> lignesCommande = new HashMap<Produit, Integer>() ;
```

- Ajouter la méthode getInstance() :

```
public final static Commande getInstance(){
        //Le "Double-Checked Singleton"/"Singleton doublement vérifié" permet
        //d'éviter un appel coûteux à synchronized,
        //une fois que l'instanciation est faite.
        if (Commande.instance == null) {
            // Le mot-clé synchronized sur ce bloc empêche toute instanciation
            // multiple même par différents "threads".
            // Il est TRES important.
            synchronized(Commande.class) {
                if (Commande.instance == null) {
                    Commande.instance = new Commande(new java.util.Date().getTime());
                }
            }
        }
        return Commande.instance;
    }
```

- Modifier le constructeur de la classe Commande comme suit :

```
private Commande( Long dateCommande) {

    this.dateCommande = dateCommande;

}
```

- Relancer l'application afin de corriger les éventelles erreurs.

1. Implémentation de l'activité correspondante

Il faut créer une commande au lancement de l'application (au niveau du contrôleur).

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
   //
    commandeEnCours = Commande.getInstance();
    //
    SimpleDateFormat df = new SimpleDateFormat( pattern: "dd-MM-yyyy");
    //
    String formattedDate = df.format(commandeEnCours.getDateCommande());
```

- Modifier la classe MainActivity comme ci-dessus et **commenter !!!**
- Lancer l'application.

Afin de réaliser des tests nous allons ajouter deux produits à la commande.

- **Dans le constructeur de la commande**, copier ces lignes de code :

```
 //création d'une catégorie
Categorie bonbons = new Categorie("bon","bonbons");
 //ajout de lignes à la commande
lignesCommande.put(new Produit("B004","Bonbons caramel Lot 4 Kg","",(float)5.0,bon-
bons),10);
 lignesCommande.put(new Produit("B005","Bonbons acidulés Lot 5 Kg","",(float)1.0,bon-
bons),2);
```

- **Revenir dans l'activité** et afficher le montant de la commande dans un toast :

```
//
Toast toast =  Toast.makeText(getApplicationContext(), text: "Affichage test de la commande "
        +formattedDate+ " montant de la commande "+String.valueOf(commandeEnCours.getTotalCommande()),Toast.LENGTH_LONG);

toast.show();
```

La suite au prochain épisode….