Contexte : Vanille
Les principales fonctionnalités sont :

- 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.

- Lancer Android Studio en tant qu'**Administrateur** (toujours).
- Après chargement complet de votre projet, lancer votre émulateur ou brancher votre téléphone.
- Ouvrir le fichier FactureActivity.java.

I. Enregistrement des informations du client

Actuellement le client doit saisir ses coordonnées pour chaque nouvelle commande, cela est fastidieux. Afin d'éviter ce désagrément à l'utilisateur nous allons enregistrer les informations saisies dans un fichier binaire pour pouvoir les récupérer ultérieurement.

La **sérialisation** est un procédé qui permet de rendre un objet ou un graphe d'objets persistant pour stockage ou échange et vice versa. Cet objet est mis sous une forme sous laquelle il pourra être reconstitué à l'identique. Ainsi il pourra être stocké sur un disque dur ou transmis au travers d'un réseau pour le créer dans un autre environnement.

A. Création de la classe de sérialisation

- Créer un nouveau package, nommé outils.
- Créer une classe Java de type Abstraite nommée Serializer dans ce package.

Cette classe contiendra 2 méthodes statiques :

- Une méthode permettant de sérialiser un objet (enregistrer un objet dans un fichier).
- Une méthode permettant de reconstituer un objet à partir d'un fichier.

Cette classe pourra être utilisée dans n'importe quel projet.

- Recopier le code des 2 méthodes et commenter :

```
public abstract class Serializer {

    /**
     * @param filename
     * @param object
     * @param context
     */
    public static void serialize(String filename, Object object, Context context) {
        try {
            FileOutputStream file = context.openFileOutput(filename, Context.MODE_PRIVATE);
            ObjectOutputStream oos;
            try {
                oos = new ObjectOutputStream(file);
                oos.writeObject(object);
                oos.flush();
                oos.close();
            } catch (IOException e) {
                //erreur de serialisation
                e.printStackTrace();
            }

        } catch (FileNotFoundException e) {
            //fichier non trouvé
            e.printStackTrace();
        } catch (IOException e) {
            //toutes les autres exceptions
            e.printStackTrace();
        }
    }

    /**
     *
     * @param filename
     * @param context
     * @return
     */
    public static Object deSerialize(String filename, Context context) {
        Object object = null;
        try {
            FileInputStream file = context.openFileInput(filename);
            ObjectInputStream ois;
            try {
                ois = new ObjectInputStream(file);
                try {
                    object = ois.readObject();
                    ois.close();
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                }
            } catch (StreamCorruptedException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return object;
    }
}
```

B. Sérialisation du client

La classe client doit implémenter l'interface Sérialisable.

```
public class Client implements Serializable {
    private String nomClient;
```

- Importer la bibliothèque correspondante.

Nous allons gérer les opérations de création et de récupération du client dans un package repository.

- Créer le package.
- Créer une classe abstraite ClientRepository.
- Ajouter un attribut statique afin de définir le nom du fichier qui contiendra le client.

```
private static final String nomFichier = "saveClient";
```

- Créer le getter de cet attribut.
- Recopier les 2 méthodes et commenter.

```
public static void EnregistrerLeClient(Client leClient, Context  context){
    Serializer.serialize(nomFichier, leClient,  context);
}

public static Client RecupererLeClient(Context context){
    return (Client)Serializer.deSerialize(nomFichier,context);
}
```

C. Gestion de l'activité FactureActivity : utilisation de la sérialisation

A la création de l'activité il faut récupérer les informations du client s'il existe :

- Recopier et Compléter la méthode recupClient( ) ci-dessous dans FactureActivity :

```
private void recupClient(){
    if (null != Serializer.deSerialize(ClientRepository.getNomFichier(), context: this)){
        Client leClient = (Client)Serializer.deSerialize(ClientRepository.getNomFichier(), context: this);
        editNom.setText(leClient.getNomClient());
        editPrenom.setText(leClient.getPrenomClient());
```

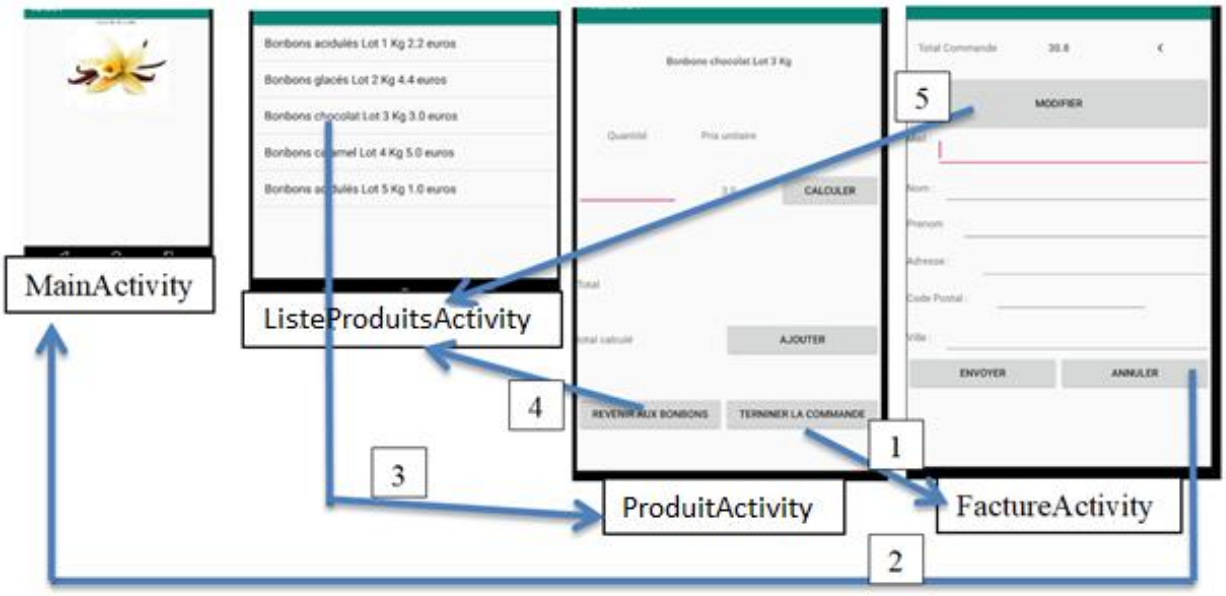
➢ Appeler la méthode à la création de l'activité : Ajouter ces 2 lignes dans la méthode init().

```
final Context context = this;
recupClient();
```

➢ Modifier la méthode bt_envoyer.setOnClickListener() afin de créer le client en sérialisant celui-ci.
➢ Tester.

II. Communication entre 2 activités : les Intentions

Notre application aura au final 4 écrans.
Voici les liaisons entre les activités que nous allons gérer dans un premier temps :



A. Intention (1) : un clic sur le bouton « Terminer la commande »

➢ Ajouter les 2 boutons manquants au layout Activity_produit.xml .
➢ Ajouter les 2 attributs correspondants à la classe java ProduitActivity.
➢ Dans la méthode init( ) faire les liens.

- Ecrire la méthode suivante qui permet de gérer le clic sur le bouton (commenter):

```
btTerminer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(view.getContext(),FactureActivity.class);
        startActivity(intent);
        finish();
    }
});
```

Cette méthode permet de faire la « transition » entre les 2 activités.

- Tester.

B. Intention (2) : un clic sur le bouton « Annuler »

Lorsque le client annule la commande l'application doit vider la liste des bonbons et revenir à l'écran de lancement de l'application.

- A vous de gérer.

C. Intention (3) : Sélection d'un produit dans la liste

1. Chargement du catalogue.

Nous allons pour l'instant simuler le chargement du catalogue dans la classe CatalogueRepository.

- Ajouter la classe CatalogueRepository fournie dans le package repository.

2. Création de l'activité ListeProduitsActivity

a) Création de la vue

- Dans le package controller ajouter une activité vide, cocher Launcher activity.
- Ouvrir le fichier activity_liste_produits.xml et ajouter une ListView (Legacy).
- Renommer la ListView.

b) Implémentation de la vue

- Ouvrir la classe ListeProduitsActivity.

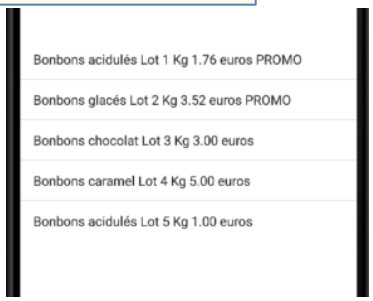
- Ajouter 2 attributs privés : une liste de produits et une ListView.

```
public class ListeProduitsActivity extends AppCompatActivity {

    private ListView lV_produits;
    private List<Produit> lesProduits = new ArrayList<>();

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

- Faire le lien avec le layout dans la méthode init( ) :

```
private void init() {

    lV_produits = findViewById(R.id.ListeProduits_LV);
    lesProduits = CatalogueRepository.RecupererLeCatalogue( context: this);
    ArrayAdapter<Produit> itemsAdapter = new ArrayAdapter<Produit>( context: this, android.R.layout.simple_list_item_1, lesProduits);
    lV_produits.setAdapter(itemsAdapter);
```

- Tester .



c) *Gestion de la sélection d'un item*

L'action du clic sur un bonbon de la liste doit déclencher l'ouverture de l'activité ProduitActivity avec le bonbon sélectionné.

Pour ce faire, cet évènement doit créer une « intention » qui ouvre l'activité, tout en conservant la position (index dans la liste) de l'item sélectionné.

Dans l'activité de départ : classe ListeProduitsActivity

- Recopier le code suivant dans la méthode init( ) et commenter.

```
lV_produits.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {
        Intent intent = new Intent( packageContext: ListeProduitsActivity.this, ProduitActivity.class);
        intent.putExtra( name: "index",position);
        startActivity(intent);
    }
});
```

Dans l'activité d'arrivée : classe ProduitActivity

- Mettre en commentaires les lignes simulant la sélection.
- Recopier le code suivant afin de modifier la méthode init( ) et commenter.

```
//récupération du Produit sélectonné
Bundle b = getIntent().getExtras();
if(b != null){
    int pos = b.getInt( key: "index");
    produitSelectionne = CatalogueRepository.RecupererLeCatalogue( context: this).get(pos);
}
```

- Tester.

Dans le constructeur de la classe commande

- Supprimer les lignes de commandes créer pour les tests et vérifier que votre application fonctionne correctement.

Pour vous aider : penser à utiliser Logcat afin de visualiser votre commande.

D. Intention (4 et 5) : retour à la liste bonbons

- A vous de jouer…