Hannachi Thamer l3 dsi EAD

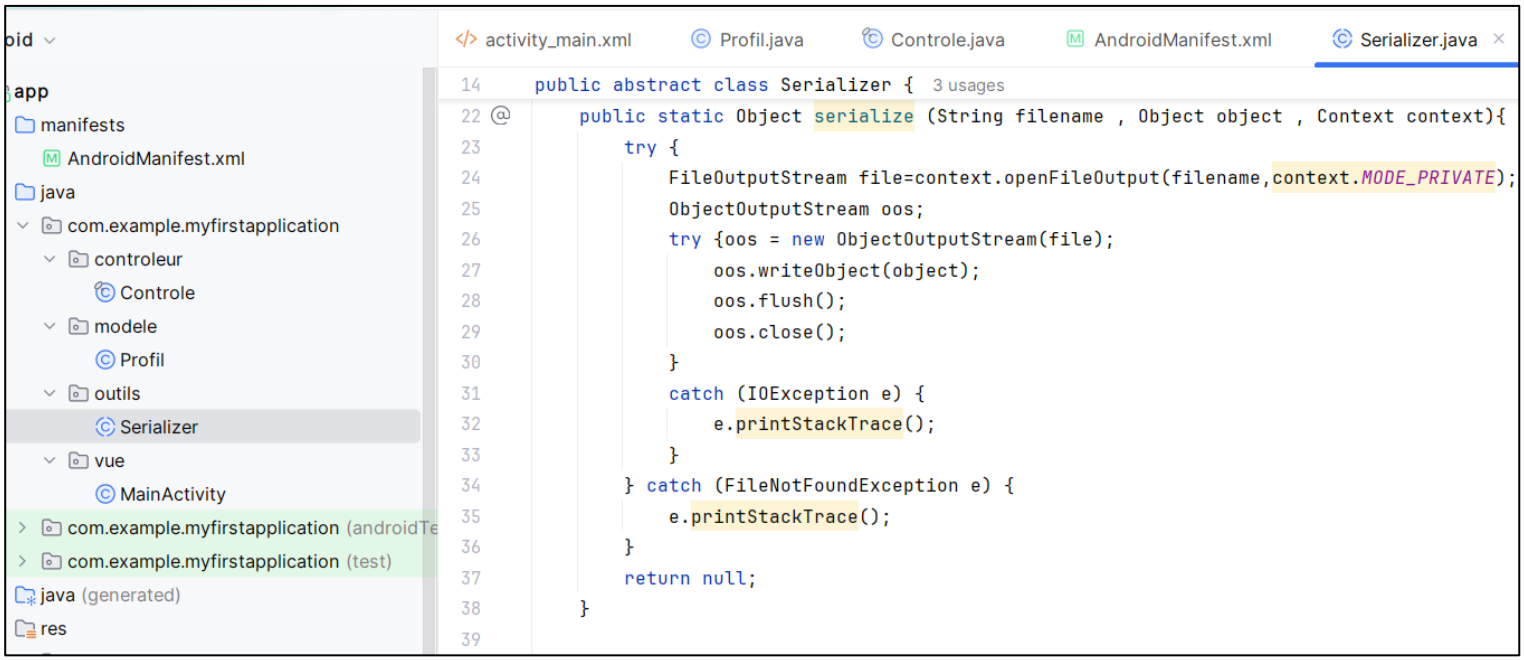
Atelier Serialisation

utilise le **principe de la sérialisation** pour sauvegarder et restaurer les données de manière persistante dans le stockage interne de l'appareil. Le tout sera organisé en suivant l'**architecture MVC (Modèle-Vue-Contrôleur)**, une approche qui sépare la logique métier, la gestion des données et l'interface utilisateur pour une meilleure modularité et maintenabilité.

pour sauvegarder des objets dans un fichier afin de les récupérer plus tard .. o utilise les methodes de Class abstraite Serializer qui permet de **sérialiser un objet Java** dans un fichier stocké dans l'espace de stockage interne de l'application Android:

1. **Signature de la méthode** :
    - serialize est une méthode statique qui accepte trois paramètres :
        - filename : le nom du fichier où l'objet sera sérialisé.
        - object : l'objet à sérialiser.
        - context : le contexte Android nécessaire pour accéder aux fonctionnalités de stockage interne.
2. **Ouverture du fichier** :
    - La méthode utilise context.openFileOutput(filename, Context.MODE_PRIVATE) pour ouvrir un fichier en mode privé. Cela signifie que le fichier sera accessible uniquement par l'application.
3. **Création d'un flux d'objets** :
    - Un objet de type ObjectOutputStream est créé à partir du flux de fichier (FileOutputStream). Ce flux permet d'écrire des objets dans le fichier.
4. **Écriture de l'objet dans le fichier** :
    - oos.writeObject(object) écrit l'objet sérialisé dans le fichier.
    - oos.flush() force l'écriture des données restantes dans le fichier, et oos.close() ferme le flux pour libérer les ressources.
5. **Gestion des exceptions** :
    - Si une erreur survient lors de l'ouverture du fichier ou de la sérialisation, les exceptions (FileNotFoundException ou IOException) sont interceptées et leur trace est affichée à l'aide de e.printStackTrace().

6. **Valeur de retour** :
    - La méthode retourne toujours null. Cela pourrait être amélioré pour indiquer si l'opération a réussi ou non.



Deserialisation :

1. **Signature de la méthode** :

deserialize est une méthode statique qui accepte deux paramètres :

filename : le nom du fichier contenant l'objet sérialisé.

context : le contexte Android utilisé pour accéder aux fichiers internes.

2. **Ouverture du fichier** :

context.openFileInput(filename) ouvre un fichier existant en mode lecture (FileInputStream).

3. **Création d'un flux d'objets** :

ObjectInputStream est créé à partir du flux de fichier (FileInputStream). Ce flux permet de lire des objets sérialisés.

4. **Lecture de l'objet** :

ois.readObject() lit et désérialise l'objet depuis le fichier.

L'objet est ensuite retourné après la fermeture du flux (ois.close()).

5. **Gestion des exceptions** :

Les exceptions suivantes sont gérées avec e.printStackTrace() pour débogage :

FileNotFoundException : Si le fichier spécifié n'existe pas.

StreamCorruptedException : Si le flux est corrompu.

IOException : Si une erreur d'entrée/sortie survient.

ClassNotFoundException : Si la classe de l'objet désérialisé n'est pas trouvée.

6. **Valeur de retour** :

Si tout se passe bien, l'objet désérialisé est retourné.

En cas d'erreur, null est retourné.

```
    public static Object deserialize (String filename , Context context){  1 usage
        try {
            FileInputStream file = context.openFileInput(filename);
            ObjectInputStream ois;
            try {
                ois =new ObjectInputStream(file);
                try {
                    Object object=ois.readObject();
                    ois.close();
                    return object;
                } catch (ClassNotFoundException e) {e.printStackTrace();}
            }
            catch (StreamCorruptedException e) {e.printStackTrace();}
            catch (IOException e) {e.printStackTrace();}


        } catch (FileNotFoundException e) {e.printStackTrace();}

        return null;
    }
}
```

Hannachi Thamer l3 dsi EAD
En déclarant classe Profil qui implémente l'interface Serializable, pour que ses instances soient sérialisées et désérialisées:

Pourquoi Serializable ?

- L'interface Serializable est une **marqueur** (marker interface). Elle ne contient aucune méthode, mais indique à la JVM que les objets de cette classe peuvent être convertis en une séquence d'octets pour être stockés ou transmis.
- Ces octets peuvent ensuite être retransformés en objets grâce à la désérialisation.

```
2
3   import java.io.Serializable;
4   import java.util.Date;
5
6   public class Profil implements Serializable {  10 usages
7
8       // constantes statiques
9       private static final Integer minFemme=15;  1 usage
10      private static final Integer maxFemme=30;  1 usage
11      private static final Integer minHomme=10;  1 usage
12      private static final Integer maxHomme=25;  1 usage
13      // proprietes
14      private Date dateMesure;  2 usages
15      private Integer poids;  3 usages
16      private Integer taille;  3 usages
```

nomFic est le nom du fichier de sauvegarde des données liées au Profil.

```
6
7   public final class Controle {  8 usages
8
9       private static Controle instance=null;  3 usages
10      private static Profil profil;  13 usages
11      private static String nomFic="saveprofil";  2 usages
12
13
```

Sauvegarde de l'objet Profil (Sérialisation) :

- La méthode creerProfile permet de créer un objet Profil et de le sauvegarder en utilisant la classe Serializer :

```
/**
 * creation de profil
 * @param poids
 * @param taille est en centimetre
 * @param age
 * @param sexe 1 pour homme et 0 pour femme
 */
public void creerProfile(Integer poids, Integer taille, Integer age, Integer sexe , Context contexte){  1 usage
    profil=new Profil(poids, taille, age, sexe);
    Serializer.serialize(nomFic,profil,contexte);
}
```

La méthode Serializer.serialize est appelée pour convertir cet objet en une séquence d'octets et le stocker dans le fichier nommé saveprofil.

Rcupération de l'objet Profil (Désérialisation) :

- La méthode recupSerialize est utilisée pour récupérer un objet Profil précédemment sauvegardé :

```
/**
 * recperation de l'objet serialisé le profil
 * @param contexte
 */
private static void recupSerialize(Context contexte) {  1 usage
    profil =(Profil) Serializer.deserialize(nomFic,contexte);

}
```

- La méthode Serializer.deserialize lit le fichier saveprofil, désérialise les octets stockés et reconstruit un objet Profil.
- L'objet désérialisé est ensuite stocké dans le champ statique profil.

Lors de l'initialisation de l'instance de la classe Controle (dans getInstance), la méthode recupSerialize est appelée pour tenter de charger un Profil déjà existant :

```
public static final Controle getInstance(Context contexte) {
  if (Controle.instance == null) {
    Controle.instance = new Controle();
    accesLocal = new AccesLocal(contexte);
    recupSerialize(contexte);
  }
  return Controle.instance;
}
```

```
    /**
     * recuperation de profil
      */
private void recupprofil(){  1 usage
    if (control.getPoids() != null) {PoidsText.setText(control.getPoids().toString());
                                     TailleText.setText(control.getTaille().toString());
                                     AgeText.setText(control.getAge().toString());
                                    if (control.getSexe()==1){radioHomme.setChecked(true);}
                                    //simule le click sur button clique
        ((Button)findViewById(R.id.calculIMC)).performClick();
    }
}
}
```

Analyse détaillée de la méthode recupprofil :

Étape 1 : Vérification de l'existence des données

```
if (control.getPoids() != null)
```

- La méthode vérifie si les données du profil ont été restaurées en mémoire, c'est-à-dire si le champ profil dans la classe Controle n'est pas null.
- Cela implique que :
    - Un objet Profil a été désérialisé à partir du fichier "saveprofil".
    - La méthode recupSerialize dans Controle a été appelée (via getInstance).

Étape 2 : Récupération des données du profil

```
{PoidsText.setText(control.getPoids().toString());
 TailleText.setText(control.getTaille().toString());
 AgeText.setText(control.getAge().toString());
if (control.getSexe()==1){radioHomme.setChecked(true);}
```

- Les données désérialisées de l'objet Profil sont utilisées pour :
    - Remplir les champs de l'interface utilisateur (PoidsText, TailleText, AgeText).
    - Cocher le bouton radio approprié (radioHomme ou son alternative implicite pour "femme").
- Les méthodes comme getPoids, getTaille, etc., accèdent directement à l'objet Profil désérialisé.

Étape 3 : Simulation d'un clic sur un bouton

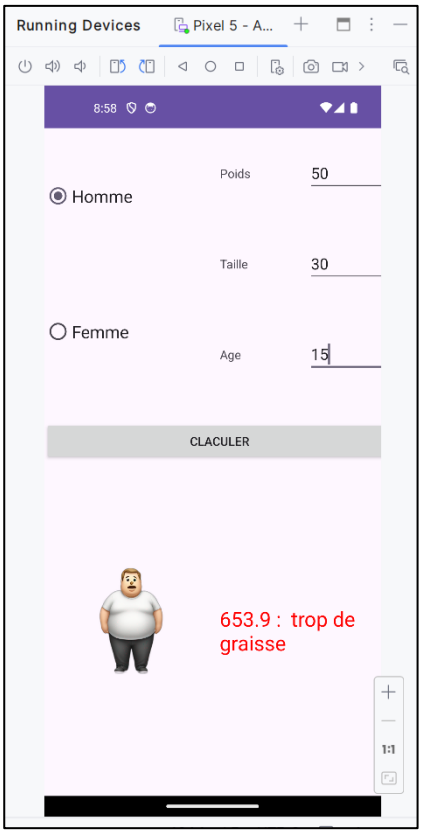
```
((Button)findViewById(R.id.calculIMC)).performClick();
```

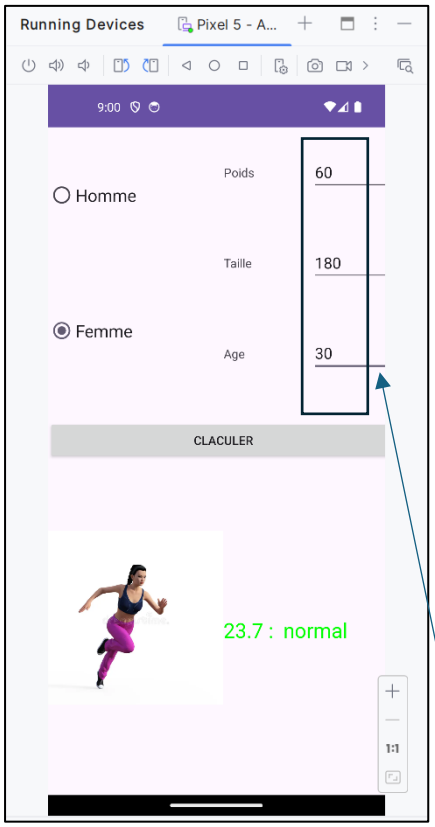
- **Action simulée :**
  - Le code simule un clic sur le bouton calculIMC.
  - Cela déclenche la méthode liée au bouton qui effectue un calcul basé sur les données du profil .
- **Lien avec la sérialisation :**
  - Cela permet de réutiliser les données restaurées immédiatement, sans nécessiter d'intervention manuelle de l'utilisateur.

Résumé :

- La méthode recupprofil tire parti de la sérialisation pour :
  1. Charger un objet Profil sauvegardé.
  2. Afficher ses données dans l'interface utilisateur.
  3. Simuler un clic pour réutiliser les données (ex., recalculer l'IMC).
- **Avantage clé :** La sérialisation permet de restaurer l'état de l'application à partir des données précédemment sauvegardées, évitant à l'utilisateur de saisir les informations à chaque lancement.

Hannachi Thamer l3 dsi EAD





Les derniers données sont recuperes