420-V41-SF
PROGRAMMATION D'APPLICATIONS MOBILES I
ÉVALUATION FINALE À CARACTÈRE SYNTHÈSE – PARTIE 4
L'ACTIVITÉ DE GESTION DES PRÉFÉRENCES
PONDÉRATION : 10%

OBJECTIFS PÉDAGOGIQUES

Ce travail vise à familiariser l'étudiante ou l'étudiant avec les objectifs suivants :

- 016T- Appliquer une approche de développement par objets.
- 017D- Concevoir et développer une application hypermédia dans des réseaux internes et mondiaux.
- 0177- Assurer la qualité d'une application

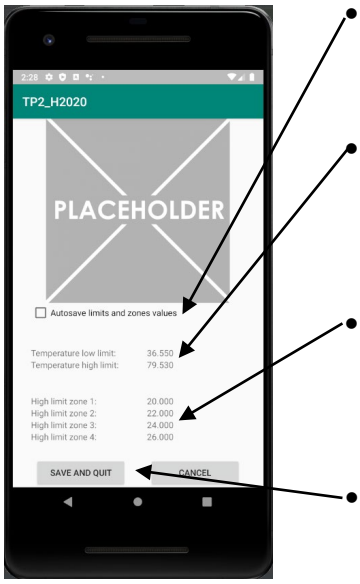
De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de documentation et de programmation.

MISE EN CONTEXTE ET MANDAT

Pour la quatrième partie de ce dernier travail, vous devez réaliser une activité Android qui permet de gérer les préférences de l'application. Pour les besoins de ce travail, il y aura 7 préférences à gérer.

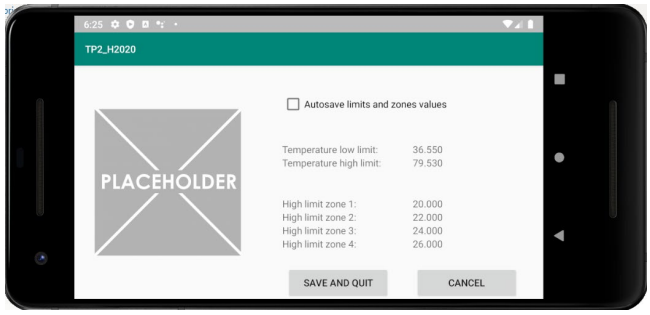
CONTRAINTES DE RÉALISATION

Pour la partie 4 du travail, vous devez produire une application Android permettant de gérer les préférences suivantes :



- Il doit être possible de choisir de sauvegarder les limites de température ainsi que les limites des zones 1@4 pour l'humidité (rappel : la limite de la zone 5 est toujours la valeur maximale)
- Les limites haute et basse de température doivent être affichées mais ne peuvent pas être éditées. La seule manière de les changer est d'activer la sauvegarde automatique et d'aller les modifier dans l'activité de température.
- Les limites hautes des zones 1@4 doivent être affichées mais ne peuvent pas être éditées. La seule manière de les changer est d'activer la sauvegarde automatique et d'aller les modifier dans l'activité d'humidité.
- Pour terminer l'activité, l'utilisateur peut soit choisir de sauvegarder sa modification ou quitter sans sauvegarder. Il retourne alors à l'activité principale.

- Vous devez adapter vos activités de température et d'humidité afin que les valeurs soient lues dans la base de données et utilisées dans les graphiques lors du chargement.
- Vous devez adapter vos activités de température et d'humidité afin que les valeurs soient sauvegardées dans la base de données lors de la fermeture des activités concernées si cette option est activée dans la base de données.
- L'activité de préférences doit supporter les changements d'orientation :



- L'activité doit supporter les changements de langue.
- Votre code doit être organisé avec le patron de conception MVC.
- L'accès à la base de données doit être organisé avec le patron de conception Repository (voir Annexe 1).
- Vous devez <u>obligatoirement</u> :
    1) Intégrer à votre projet l'interface `Repository` donnée avec cet énoncé et implanter **<u>toutes</u>** ses méthodes.
    2) Rajouter la méthode suivante à l'interface `View` :
        - `PreferencesRepository getRepository();`
- Vous devez réaliser les tests unitaires de la classe `PreferencesRepository` qui permettent l'accès aux données des préférences. Ces tests unitaires feront l'objet d'une remise partielle (voir modalités de remise). Afin de pouvoir créer votre base de données, vous avez besoin d'une instance de la classe `Context`. Lors de l'exécution « normale » de votre application, le contexte provient de l'activité. Or, vous n'avez pas accès à l'activité lors d'un test JUnit standard. Vous devrez donc utiliser un « test instrumenté ». Vous aviez utilisé de tels tests dans le contexte des tests d'interface automatisés. Heureusement, on ne vous demande pas ici de tester l'interface mais seulement d'utiliser le contexte qui vous est accessible.

  <u>Note importante</u> : Il est possible de créer une base de données dite « en mémoire » en informant le `ApplicationDatabaseHelper` que le nom de la base de données est `null`. Ceci vous sera d'une grande utilité pour les tests unitaires.
- Vous serez évalués(ées) de la manière suivante : 1) Complétion de la suite de tests et 2) Réalisation d'une suite de tests que nous fournirons (d'où l'importance de respecter la signature des méthodes spécifiées juste avant).

- Vous devez commenter le code en mode javadoc. La forme est ce qui est le plus important, mais le fond doit décrire avec justesse ce que vous faites et ce, **en français**. Vous devrez générer et remettre votre javadoc.

L'opérationnalité des fonctionnalités ne garantit pas la cote de passage. L'architecture et la qualité du code réalisé ainsi que des documents complet et professionnel sont primordiaux. Si le professeur le juge à propos, toute étudiante ou tout étudiant pourra être convoqué à une rencontre d'évaluation sur Teams pour vérifier son degré d'acquisition des connaissances et d'appréhension de la solution proposée.

CONTEXTE DE RÉALISATION ET DÉMARCHE DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **seul**.
Les pages qui suivent résument les étapes du projet ainsi que les biens livrables devant être remis à la fin de chacune d'elles.

Étape 1 : Analyse de la situation

Dans cette étape, vous devez prendre connaissance du mandat qui vous est confié.

Biens livrables :

- Aucun.

Méthode:

- s/o

Étape 2 : Programmation de l'application

Vous devez procéder à la codification des fonctionnalités demandées ainsi que des tests spécifiés.

Biens livrables :

- Code source de l'application documenté (javadoc). Vous devez aussi générer la javadoc dans un dossier à part.

Méthode:

Ce travail fera l'objet de **2 remises** :

Remise partielle :

- L'implantation de toutes les méthodes du `Repository` et la suite de tests unitaires complète du `Repository` devront être remise dans une archive .zip identifiée en fonction de votre nom. Par exemple, pour l'étudiant nommé Pierre Poulin, l'archive devrait être nommée `EFCS3_Repository_PierrePoulin.zip`.

Note : Cette archive est l'archive complète du projet et peut contenir du code en développement pour le reste de l'application. Cependant, le code remis doit compiler et contenir l'implantation finale (code et javadoc) ainsi que les tests unitaires du `Repository`. Celui-ci fera l'objet d'une correction finale sur réception.

Cette archive doit être déposée sur LÉA **avant le 14 mai 2020 13h**.

Remise finale :

- Suivant la réception de la remise partielle, la suite de tests unitaires du professeur sera rendue publique. Celle-ci devra être incluse dans votre projet et les ajustements potentiels devront être faits afin de la rendre fonctionnelle. Celle-ci sera exécutée à la correction finale. Les éléments de la remise partielles ne seront pas recorrigés.
- Vous devez remettre votre code source dans une archive .zip identifiée en fonction de votre nom. Par exemple, pour l'étudiant nommé Pierre Poulin, l'archive devrait être nommée `EFCS3_PierrePoulin.zip`.
- Vous devez aussi remettre le fichier `RemiseEFCS – Partie 4.docx` accompagné de la grille d'auto-évaluation correctement complétée.

Cette archive doit être déposée sur LÉA **avant le 18 mai 2020 8h**.

MODALITÉS D'ÉVALUATION

Vous trouverez dans le fichier `RemiseEFCS – Partie 4.docx` la grille de correction qui sera utilisée pour le travail. **Cette grille indique la pondération accordée à chacune des parties du projet**.

- Tous les **biens livrables** de chacune des étapes devront être **remis à temps** et selon les modalités spécifiées.

ANNEXE 1 – LE PATRON DE CONCEPTION REPOSITORY SOUS ANDROID

Le patron `Repository` est utile pour bien structurer la couche d'accès aux données. Son principe est simple : une classe de repository pour chaque table de la base de données permettant d'obtenir, d'ajouter, de mettre à jour et de supprimer des enregistrements. Ceci nous permet de centraliser les requêtes et de découper la logique d'accès aux données de la structure MVC.

Afin d'établir le contrat de la couche d'accès aux données, l'interface `Repository` est donnée avec cet énoncé et doit être implantée dans un package à part (le `Repository` est souvent perçu comme un service). C'est une interface gabarit (template) qui contient les méthodes suivantes :

```
public interface Repository<T>
{
    boolean insert(T data);
    T find(long id);
    T findLast();
    boolean save(T myObject);
    boolean delete(T myObject);
    boolean delete(long id);
}
```

Dans le cadre de ce travail, nous aurons besoin d'un `PreferencesRepository` qui, bien entendu, transigera avec la table `Preferences` et manipulera « l'objet de données » correspondant à cette table. Un repository obtient lors de sa construction la base de données pour pouvoir effectuer son travail.

La création du `Repository` doit donc être effectuée par les activités qui en ont besoin (un repository peut être utilisé par plusieurs activités, nous en avons ici un bel exemple) car les activités ont la capacité de créer une base de données et de facilement gérer la connexion à l'aide du cycle de vie.