# 420-V41-SF
# PROGRAMMATION D'APPLICATIONS MOBILES I
# ÉVALUATION FINALE À CARACTÈRE SYNTHÈSE – PARTIE 2
# L'ACTIVITÉ DE TRAITEMENT DES DONNÉES DE TEMPÉRATURE
# PONDÉRATION : 12%

## 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

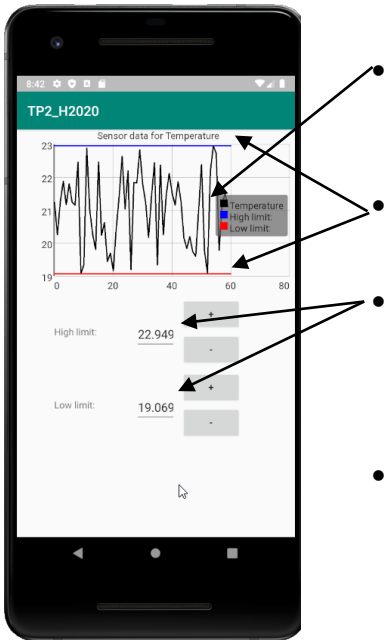
De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de documentation et de programmation.

## MISE EN CONTEXTE ET MANDAT

Pour la seconde partie de ce dernier travail, vous devez réaliser une activité Android qui permet de traiter les données de température acquises dans la première partie de ce travail. Veuillez noter qu'il vous est toujours possible de simuler les données dans l'éventualité où votre première partie n'était pas totalement fonctionnelle.

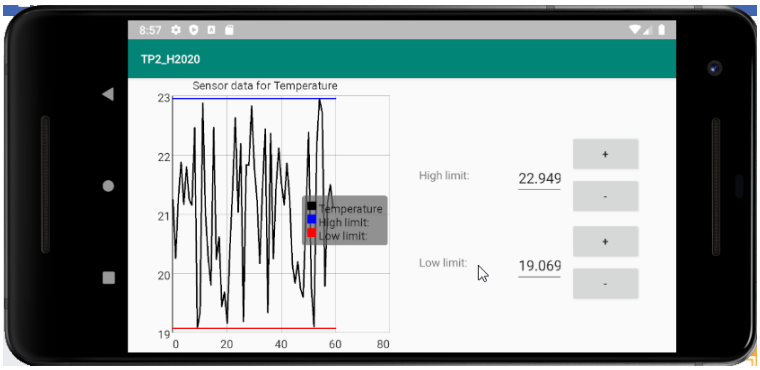
## CONTRAINTES DE RÉALISATION

Pour la partie 2 du travail, vous devez produire une application Android offrant les fonctionnalités suivantes :




- L'application doit afficher un graphique des températures observées. Pour cela, utilisez la librairie `com.jjoe64:graphview:4.2.2`.
- L'application doit permettre de spécifier des limites hautes et basses pour lesquelles une alarme pourrait être signalée.
- Il doit être possible de spécifier dynamiquement ces limites ou de les augmenter/diminuer à l'aide de boutons appropriés. Les modifications alors apportées à la limite concernée sera alors un « pas » calculé ainsi : ($température_{max}$-$température_{min}$)/10.
- En aucun cas, la limite inférieure ne peut être plus grande que la limite supérieure. Advenant cette possibilité, les boutons appropriés devront être désactivés.

- L'activité de température 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.

**NOTE IMPORTANTE**

Nous sommes conscients qu'il se peut que la partie 1 n'ait pas été complétée en totalité par tout le monde. Afin de ne pas pénaliser indûment pour la suite des choses celles ou ceux qui ne l'auraient pas terminée, nous vous fournissons les classes `SensorData` et `SensorValue` fonctionnelles. La classe `SensorData` représente toutes les données que vous aurez récupérées du service Web pour un capteur donné tandis que la classe `SensorValue` représente une seule donnée.

Vous **devez** les utiliser pour la suite du travail. Cela sera nécessaire afin d'harmoniser le code entre les personnes, afin de faciliter le débogage à distance ainsi que pour uniformiser notre correction. Vous devrez donc modifier votre tâche d'accès aux données pour retourner désormais une instance de `SensorData` à votre activité principale.

Le constructeur de votre modèle devra donc prendre une instance de `SensorData` en paramètre.

- Vous devez réaliser les tests unitaires du **contrôleur**. A cette fin, vous devrez obligatoirement utiliser les méthodes suivantes :
    - Modèle :
        - `void setHighLimit(double newHighLimit);`

          Cette méthode permet de modifier la limite supérieure.
        - `double getHighLimit();`

          Cette méthode permet d'obtenir la limite supérieure.
        - `void setLowLimit(double newLowLimit);`

          Cette méthode permet de modifier la limite inférieure.
        - `double getLowLimit();`

          Cette méthode permet d'obtenir la limite inférieure.

- Contrôleur :
  - `void onHighLimitDown();`

    Cette méthode est appelée lorsque l'utilisateur tente de diminuer la limite supérieure dans la vue. La logique associée devra évidemment être testée.
  - `void onHighLimitUp();`

    Cette méthode est appelée lorsque l'utilisateur tente d'augmenter la limite supérieure dans la vue. La logique associée devra évidemment être testée.
  - `void onLowLimitUp();`

    Cette méthode est appelée lorsque l'utilisateur tente de diminuer la limite inférieure dans la vue. La logique associée devra évidemment être testée.
  - `void onLowLimitDown();`

    Cette méthode est appelée lorsque l'utilisateur tente d'augmenter la limite inférieure dans la vue. La logique associée devra évidemment être testée.

- Interface `TemperatureView` :
  - `void update();`

    Cette méthode est appelée pour mettre à jour les informations affichées dans la vue en fonction des données contenues dans le modèle.
  - `void setHighLimitDownAndLowLimitUpEnabled(boolean enabled);`

    Cette méthode est appelée pour ajuster l'accessibilité des boutons appropriés dans la vue.
  - `boolean isHighLimitDownAndLowLimitUpEnabled();`

    Cette méthode permet de vérifier l'accessibilité des boutons appropriés dans la vue.

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).

**Note** : La création du contrôleur nécessite un modèle et une vue. Vous devrez donc trouver une façon « imaginative » afin de spécifier une vue au contrôleur lors des tests.

- 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. Vous n'avez pas à commenter les classes `SensorData` et `SensorValue`.

**Méthode**:

- 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 `EFCS_Partie2_PierrePoulin.zip`.
- Vous devez aussi remettre le fichier `RemiseEFCS – Partie 2.docx` accompagné de la grille d'auto-évaluation correctement complétée.
  Cette archive doit être déposée sur LÉA **avant le 4 mai 2020 à 7h59**.

## MODALITÉS D'ÉVALUATION

Vous trouverez dans le fichier `RemiseEFCS – Partie 2.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.