Cours 420-5B6-LI **Applications monopostes II** **Automne 2022** **Cégep Limoilou** **Département d'informatique** **Professeur : Martin Simoneau**	**TP2 (18(ou 24) %)** **Application d'assistance mathématique** **Module Maven, Spring et JavaFX**

Objectifs

- Créer des modules Maven
  - Organisation du code (conception organique)
  - Mise en place des patrons de conceptions adéquats
  - Mettre l'accent sur la réutilisabilité du code
- Créer des applications *javaFX* avec *Maven*
  - *Multicontrôleurs*
  - *Multifenêtres*
  - *Conception de UI*
- Utiliser les Streams java.
- Utiliser Spring *FrameWork* / *Boot*
- Utiliser les bons patrons de conception
  - Inversion de dépendances
  - Injection de dépendances
  - Observateur
  - *AbstractFactory*
- Concevoir des algorithmes récursifs
- Tester le code avec des *Mocks*
- Faire des animations

À remettre :

- Le travail sera remis sur *Léa* et sur *BitBucket* à la date indiquée.
- Le professeur doit être ajouté dans votre équipe *BitBucket*.

Contexte :

- Remettre vos projets sur *Léa* **et** sur *BitBucket*
- Une remise par équipe

Préparation

- Vous devez être en équipe de 3 ou 4 personnes (pas plus de 4 en raison des limites *Bitbucket*)
- **IMPORTANT** : Toutes les communications importantes (discussions, décisions, propositions) de l'équipe doivent être faites dans le canal Teams créer pour votre équipe
- Nous allons continuer le TP1. On continuera donc sur le même dépôt.
  - **IMPORTANT** : Mettre un **tag** git de la version finale du TP1 avant de commencer les commits sur le TP2.

Travail à faire

- On poursuit le travail du TP1.
- Modification du module de séries pour tenir compte des catégories. Il faut modifier la banque de données et l'interface. Chaque catégorie peut avoir :
  - Plusieurs sous-catégories.

        - Plusieurs équations associées. Chaque équation appartient à une seule catégorie. Lorsqu'une équation est utilisée pour générer une série, la catégorique de l'équation est automatiquement associée à la nouvelle série.
        - Plusieurs séries de données associées. Chaque série de données appartient à une seule catégorie.
        - Une seule catégorie parente.
    - Modification du module d'édition pour mettre les équations en mémoire et pour les associer aux mêmes catégories que les séries
        - Les équations doivent être sauvegardées en mémoire et associée à une catégorie.
    - Animation d'interface utilisateur.
        - On demande à chaque personne dans l'équipe de produire une animation de l'interface pertinente, attrayante et non distrayante.
        - Il doit y avoir une animation de l'interface qui place chacune des fenêtres sur l'écran afin d'utiliser l'Espace disponible au maximum
    - Amélioration de l'UI avec le graphique pour édition interactive intuitive
        - L'utilisateur devra pouvoir modifier les données de façon le plus possible graphique à partir du *LineChart*. Les *Serie* et *Data* sont des objets *javaFX* qui peuvent être modifiés et reformatés. On peut leur ajouter des menus ou changer leur graphique.
            - En utilisant la méthode setNode() sur les *Data* on peut placer l'élément graphique que l'on veut, *Rectangle, Circle*… On pourra alors ajuster son apparence et son contenu facilement.
            - On peut également détecter les clics dans le *LineChart*, sur les données ou sur les séries.
    - Ajoutez un module de statistique qui sort des informations intéressantes. **VOUS DEVEZ PRODUIRE LES RÉSULTATS VOUS-MÊME À L'AIDE D'ÉQUATIONS RÉCURSIVES** et non avec des requêtes SQL. Voici des **exemples** intéressants:
            - Nombre d'équations total
            - Nombre de sous-catégories pour une catégorie déterminée par l'utilisateur.
            - Équations et séries qui sont dans la même catégorie.
            - Trouver tous les éléments qui ont le même nom (doublon) dans la hiérarchie
            - Toutes les séries qui ont plus ou moins de données qu'une valeur demandée par l'utilisateur.
            - …
        - Manipulations
            - :(**Obligatoire**) Créez l'arbre des catégories à partir d'un fichier texte au format:
                - Nom de catégorie séparée par des "/". Toutes les catégories parentes sont toujours indiquées à chaque ligne. Les catégories parentes doivent être saisies avant les catégories enfant. Exemple:

Categorie1

Categorie1/Categorie2

Categorie1/Categorie2/CategorieA

Categorie1/Categorie3

Categorie1/Categorie3/CategorieB

…

- (**Obligatoire**) Remplir les *TreeView* en fonction des catégories en BD à l'ouverture de la fenêtre qui contient un TreeView.
- Sortir toutes les équations ou séries dans une seule liste.
- (**Obligatoire**) Convertir l'arbre quelconque des catégories en arbre binaire dérivé et l'afficher avec un *TreeView*:
- Placez toutes les séries dans un *RedBlack* avec le nom de la série comme clé. Utilisez le *RedBlack* pour le faire des recherches rapides. Vous le ferez afficher quelque part (utilisez la méthode de conversion du formatif).
- …

- **Exigences :**
  - L'application doit avoir une fenêtre ***À propos***
    - qui contient :
      - L'année
      - Le nom du professeur
      - Le nom de votre équipe
      - Le nom du produit
      - Vos noms
      - Cégep Limoilou, Application monoposte II
  - Le patron *MVC* doit être respecté en tout temps :
    - Vous devez utiliser l'une des stratégies vues en classe pour échanger les données entre le modèle et la vue. Créez les types en conséquence.
    - Dépendances bien gérées
    - Les contrôleurs ont un minimum de code
    - Bon découpage des modules
  - Les vues doivent être intuitives et propres.
    - Il doit y avoir au moins 2 vues et la fenêtre *à propos*
    - L'une des vues doit avoir au moins un contrôleur embarqué (*multi-contrôleur*)
  - Le redimensionnement des fenêtres doit être bien géré et utilisé judicieusement.
  - L'application doit utiliser au **maximum** les *Streams* pour effectuer ses tâches.
  - Utilisation de fichiers de propriétés (avec Spring) pour configurer les éléments importants
  - Chaque développeur doit produire **ce qui est demandé dans le rapport d'activités** :

Évaluation

- Il y aura 2 parties à l'évaluation. Une évaluation individuelle et une évaluation de groupe. Chaque partie compte pour 50%:
    - Évaluation individuelle. Chaque membre de l'équipe reçoit une note individuelle pour cette partie. Seul le travail que vous avez "*comité*" sur git peut vous être attribué. Faites vos propres commits.
        - Évalué à partir de :
            - Votre participation dans le canal de communication de l'équipe.
            - Votre participation dans la réalisation des travaux. Elle est évaluée à partir de l'historique GIT. Vous êtes donc responsable de *commiter* le travail que vous avez fait vous-même.
        - Critères individuels :
            - Quantité de travail fait
            - Qualité du travail fait
                - Respect des principes SOLID
            - Variété du travail fait
                - Design
                - Programmation
                - UI
                - Gestion de projet Git
            - Avancement en continu (participation à chaque livrable)
        - Chaque personne doit démontrer qu'elle sait utiliser les *streams* et les patrons de conception (MVC, observateur, inversion de dépendance, factory,
    - Évaluation collective (**il faut obtenir au moins 60% dans l'évaluation individuelle pour obtenir la note d'équipe**).
        - Critère :
            - Atteintes des objectifs
            - Qualité générale de la conception du code
            - Qualité de la présentation (intuitif et propre)
            - IMPORTANT : **Le projet remis fonctionne directement sans modification.**

FIN