Cours 420-203-RE ***Développement de programmes dans un environnement graphique*** Automne 2023 Cégep Limoilou Département d'informatique	Tp2 Calculatrice multivariables Événements *JavaFX* Interface *FXML* (12 %)

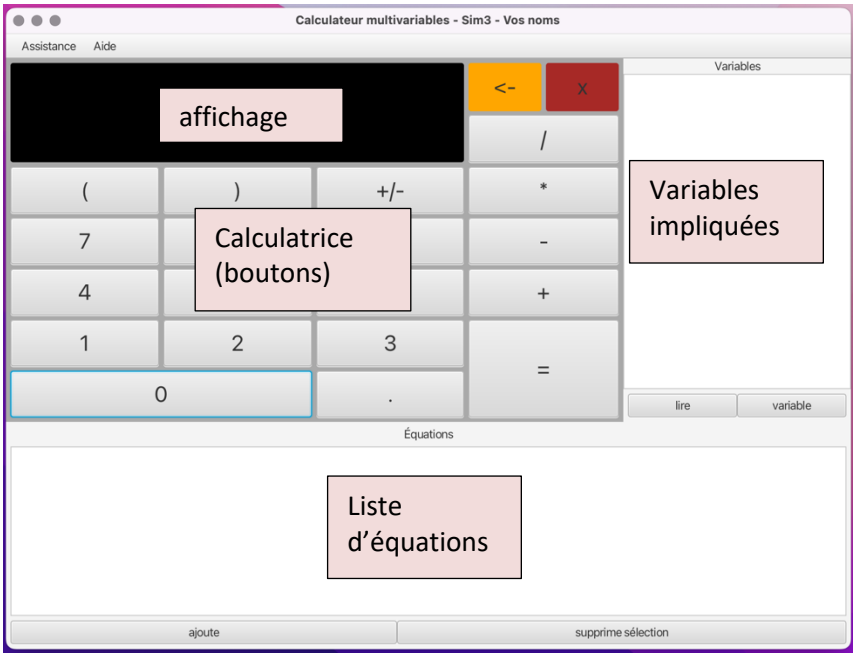
CONTEXTE DE RÉALISATION DE L'ACTIVITÉ :

- Durée : 3 semaines
- Ce travail peut être réalisé en équipe de 2 ou 3 personnes (avec l'accord du professeur)
- Suivre les consignes additionnelles sur le canal *Questions générales* de l'équipe Teams du cours.

OBJECTIFS

- Comprendre et utiliser les événements *javaFX*
- Programmer des menus, dialogues et autres éléments d'interface graphique *javaFX.*
- Lier les fonctionnalités à l'interface graphique.
- Créer un code clair et concis.
- Communiquer et partager le travail à faire.

À partir du début d'interface graphique fournie, vous allez devoir réaliser le «calculateur multivariable» suivant :



- Il s'agit d'un calculateur multivariable, c'est-à-dire qui calcule le résultat d'équations à plusieurs variables.
  - Si les équations sont saisies et effacées par l'utilisateur, les variables sont plutôt déduites directement à partir des équations. Par exemple, si l'utilisateur entre l'équation a0=b0+c0. Le système va ajouter une équation pour a0 et 2 variables pour b0 et c0. De même lorsqu'une équation est effacée, il faut effacer toutes les variables qui ne sont plus utilisées par les équations restantes.
  - Si une équation est saisie avec le même nom qu'une variable existante, l'équation remplace alors la variable existante. La variable disparaît simplement. L'équation qui devait utiliser la variable disparue devra plutôt utiliser la nouvelle équation à sa place. Ainsi le calcul d'une équation implique le calcul récursif de toutes les équations impliquées directement et indirectement.
  - Lorsqu'on efface une équation, il faut remettre la variable si l'équation était utilisée par une autre équation

Activités à réaliser

Création de l'UI :

- Vous devez mettre vos noms dans le titre de la fenêtre principale.
- Vous pouvez réaliser l'UI avec *SceneBuilder* ou par programmation, mais on vous recommande fortement de le faire avec *SceneBuilder*.
    - Indices
        - La section avec les boutons du calculateur utilise un *GridPane*
        - Le gestionnaire racine est un *BorderPane*
        - Styles à utiliser:
            - Couleur de fond: ***-fx-background-color***
            - Couleur de caractère : ***-fx-text-fill***

- Les vidéos vous montre le comportement de la fenêtre lorsqu'on la redimensionne. Vous devez le reproduire le plus fidèlement possible. Remarquez qu'on veut surtout donner de l'importance à la largeur de l'affichage.
- L'application possède 2 menus:
    - Fonctions:

      N.B. Contient une boîte à cocher
    - Aide:

- Votre application aura à communiquer avec l'utilisateur au moyen de 2 sortes de boîtes de dialogue : la boîte standard de message/avertissement/erreur et la boîte de saisie de texte. Vous pourrez également utiliser des dialogues plus avancés à votre convenance. La classe Utilitaire ***DialoguesUtils*** doit gérer tous les dialogues de l'application. Essayez de bien réutiliser le code.

Programmation des comportements :

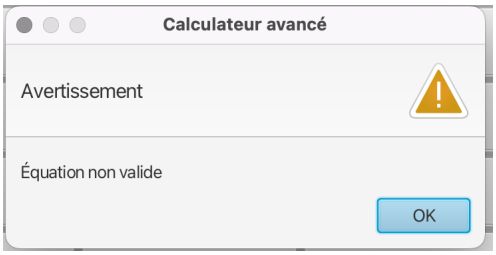
La classe ***Equation*** dans le package *modele* sert à retenir les informations pour une équation donnée. Elle est complète, mais vous pouvez y apporter des changements (à vos risques!).

- Elle contient les attributs suivants :
    - Le **nom** : le nom de l'équation (une lettre ou un mot sans espace terminé par un chiffre). Notez que les variables et les équations ont les mêmes règles de nomenclature parce qu'ils sont interchangeables.
    - Les **éléments requis** : les variables ou équations qui sont nécessaires pour effectuer le calcul de cette équation.
    - **L'expression** : la chaîne de caractères qui constitue l'équation. Ex : *sin(a0)+cos(b0)*
- Elle contient une méthode ***extraitElements*** requise qui retourne un set contenant toutes les variables et/ou équations qu'elle requiert.

Les fonctionnalités du programme sont assurées par la classe ***MoteurCalcul*** dans le package *modele*. Seule cette classe peut utiliser la librairie *mathParser.mXparser* qui est déjà configurée dans le projet qu'on vous a remis.

- Vous devez choisir les attributs appropriés pour le moteur de calcul. Vraisemblablement des collections Java qui vont contenir les variables et les équations.
- Le moteur de calcul dispose des méthodes suivantes :

- `private Set<String> determineToutesVariablesRequises() {`
  - C'est une méthode privée qui vous sera très utile pour les autres méthodes. Elle retourne un set contenant toutes les variables qui sont utilisées par l'ensemble des équations.
- `private void ajouteVariable(String variable, double valeur)`
  - Cette méthode ajoute la variable dans le moteur de calcul. Une variable est emmagasinée dans un objet ***Constant*** de *mXParser*. Vous devez pouvoir retrouver rapidement une variable en fonction de son *nom*.
- `public void setValeurVariable(String nomVariable, double valeur)`
  - donne une valeur à une variable qui est déjà emmagasinée dans le moteur de calcul.
- `public void ajouteEquation(String nouvelleEquation)`
  - Elle ajoute une équation dans le moteur d'équation. Vous devrez l'enregistrer dans l'attribut que vous aurez réservé à cette fin.
  - L'utilisateur saisi une équation sous la forme ***<nom_equation>=<expression>*** où <*expression*> est une expression mathématique contenant des variables et des expressions *mathXParser*. La méthode commence par créer une instance de *Equation* avec les informations extraites de cette chaîne de caractères.
  - L'équation doit impliquer au moins une variable sinon elle est refusée et le dialogue suivant est affiché à l'utilisateur :



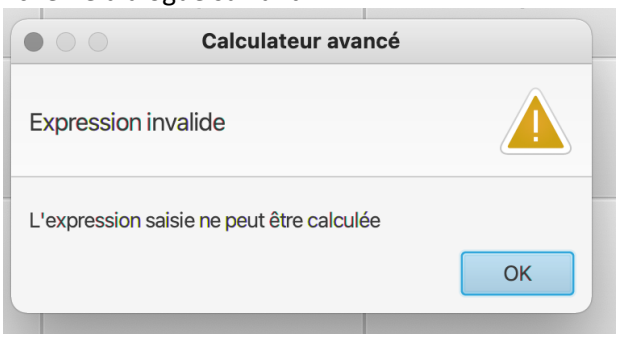
  - Si l'équation a le même nom qu'une variable existante, l'équation remplace alors cette variable. Il faudra donc supprimer la variable du même nom.
  - Si l'équation utilise une nouvelle variable, cette dernière doit être ajoutée dans le moteur de calcul. Attention, une équation peut utiliser d'autres équations. Il ne faut pas créer de variables s'il y a déjà une équation avec le nom requis. De plus, attention de ne pas écraser les variables existantes.
- `public void effaceEquation(String nomEquation)`
  - Efface l'équation avec le nom *nomEquation* dans le moteur de calcul
  - Si l'équation était utilisée par une autre équation, il faut alors ajouter une variable pour remplacer l'équation qui doit être effacée. L'équation effacée est donc remplacée par une variable du même nom.
  - Efface également toutes les variables qui ne sont plus utilisées par aucune équation. Les variables que l'équation retirée était la seule à utiliser.
- `public double calcule(String nomEquation)`
  `et`
- `public double calcule(Equation équation)`
  - Ces méthodes effectuent le calcul d'une équation de façon récursive.
  - Elle doit d'abord obtenir tous les éléments requis pour effectuer le calcul (variables et autres équations). Notez que l'équation connaît cette information.
  - Elle doit ensuite effectuer le calcul en utilisant une *Expression* de *mXParser*.
    - La classe *Expression* doit recevoir une expression et un tableau contenant des objets *Constant* pour chacune des variables utilisées dans l'expression.
    - On peut obtenir le résultat en lançant la méthode ***calculate*** de *Expression*.
  - Elle doit différencier les variables des équations. On peut obtenir directement la valeur d'une variable. La valeur d'une fonction doit être obtenue en lançant récursivement le

calcul de cette dernière jusqu'à ce que toutes les équations impliquées aient été calculées.

- Vous devez prévenir les équations à dépendances circulaires (c requiert b qui requiert c) sois en empêchant de les saisir (idéal), sois en affichant un message au moment de les utilisés (moins intéressant).

Lorsque ce n'est pas spécifié autrement, vous pouvez associer les comportements directement à partir de *SceneBuilder* .

- **Boutons principaux -** 1,2,3,4,5,6,7,8,9,0,(,),+,-,*,/ et « . » (en gris pâle)
  - Le comportement associé à ses boutons est simple, il suffit d'ajouter le texte du bouton dans l'affichage. Au fur et à mesure que l'utilisateur appuie sur les boutons, l'expression mathématique se forme dans l'afficheur. Notez que vous pouvez associer une même méthode à plusieurs composants différents dans *SceneBuilder*.
- **Bouton « = »** - égale
  - Ce bouton est très important, car il déclenche le calcul. Il doit recueillir le contenu de l'affichage puis le transmettre au moteur de calcul pour obtenir le résultat désiré. Si la valeur retournée est NaN vous devez afficher le dialogue suivant :



  - Les vidéos présentent une opération de calcul.
- **Boutons d'effacement** – "<-" et "*x*"
  - Le bouton « *x* » doit être *brown* et le bouton « <- « doit être *orange*
  - Lorsque l'utilisateur appuie sur le bouton d'effacement « <- », l'application doit effacer le dernier caractère saisi par l'utilisateur. Celui qui se trouve le plus à droite de l'affichage. Faites les manipulations nécessaires sur la chaîne de caractères contenus dans l'affichage pour obtenir ce résultat.
  - Lorsque l'utilisateur appuie sur le bouton vide, l'affichage est vidé au complet.
  - Le vidéo ***tp2-calcul.mov*** présente les comportements des 2 boutons d'effacement.
- Bouton =/-
  - Il doit changer le signe de ce qui est dans *l'affichage*. Cette fonction s'applique à l'affichage complet. La cohérence de l'opération est la responsabilité de l'utilisateur.
- **Listes**
  - **Liste d'équations**
    - Les équations sont affichées avec le format suivant :
      - *<nomVariable>=<expression>* requiert: *<listeVariablesRequises>*

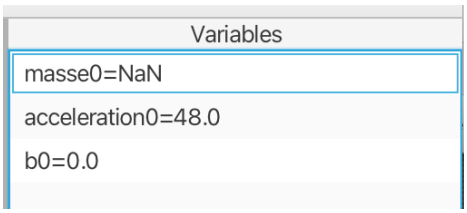
Exemples :

Équations
force0=masse0*acceleration0 requiert: [masse0, acceleration0]
a0=sin(b0)+48 requiert: [b0]

- Elle doit afficher en tout temps les équations qui sont emmagasinées dans le moteur de calcul. Une stratégie simple consiste à reconstruire cette liste complètement chaque fois qu'il y a un changement dans les équations. Notez que vous devrez probablement également rafraîchir la liste des variables si les équations changent.
- Dans le bas de la liste d'équations il y a 2 boutons :

  - Le bouton ***ajoute*** permet d'ajouter une équation dans le moteur de calcul. Il utilise le texte saisi dans l'affichage comme équation. Le texte doit avoir la forme prévue à cette fin (voir ajout d'équations dans le moteur de calcul).
  - Le bouton supprime efface l'équation qui est sélectionnée dans la liste d'équation du moteur de calcul.

- Liste de variables
  - Les variables doivent être affichées avec le format suivantL
    - <nomVariable>=<valeurVariable>
      Exemple :

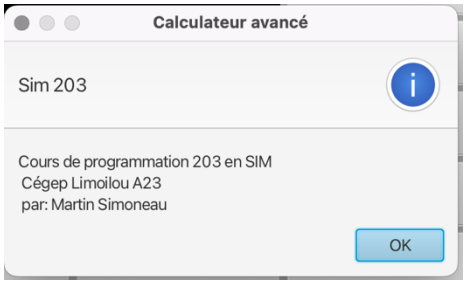


  - Elle doit afficher en tout temps les variables qui sont emmagasinées dans le moteur de calcul.
  - Dans le bas de la liste de variables vous trouver les 2 *ToggleButton* suivants :

    - Le premier bouton indique si la variable sera lue ou écrite. « Lire » signifie mettre sa valeur ou son nom dans l'affichage et « écrire » signifie prendre le contenu de l'affichage, calculer sa valeur et la mettre dans la variable du moteur de calcul correspondante.
    - Le second bouton indique si l'on veut lire (ajouter dans l'affichage) le nom de la variable ou sa valeur.
    - Le premier bouton doit avoir le texte ***lire*** lorsqu'il n'est pas enfoncé et le texte ***écrire*** lorsqu'il est enfoncé. Lorsque le premier bouton est enfoncé, le second bouton ne sert plus. Il est donc désactivé (méthode *setDisable*()). En effet c'est inutile d'écrire une variable dans une variable.
  - Lorsque l'utilisateur **double clic** sur une variable de la liste, le programme doit effectuer l'action correspondant à ce qui est prescrit par les états des 2 boutons *toggle*.

- **Menus**
  - Si le menu *Assistance/ Assistance visuelle* est activé,
    - Le texte du bouton qui se trouve sous la souris sera agrandi à 36 points au lieur de 24pts. Évidemment, lorsque la souris quitte le bouton, le texte reprend sa taille normale.
    - Pour cette fonctionnalité vous devez placer les gestionnaires nécessaires par programmation en utilisant *addEventFilter* ou *addEventHandler*.

- Le menu *aide/À propos* fait sortir un dialogue d'informations avec la description du cours et le nom des auteurs.



- **5 équations par défaut** – 5
  - Votre application doit présenter dès l'ouverture, dans la liste d'équations, les 5 fonctions suivantes :
    - *sin0=sin(x0)*
    - *cos0=cos(x0)*
    - *inverse0 =1/x0*
    - *exp0=x0^e0. (où e0 est une variable)*
    - *linear0=a0*x0+b0 (où a0 et b0 sont des variables)*
- ***Tests unitaires***
  - Chaque personne dans l'équipe doit ajouter 2 tests unitaires pertinents supplémentaires à la classe *MoteurDeCalculTest*. Les tests de chaque personne doivent tester des éléments différents.

Informations utiles

- ListView
  - Pour consulter les éléments d'un ListView
    - *List.getItems()*
  - Pour connaître l'élément sélectionné
    - *Liste.getSelectionModel().getSelectedItem();*
- *Pour séparer une chaîne de caractères*
  - Méthode *split* de la class *String*

Barème D'évaluation :

- La classe *DialoguesUtils* est la seule qui peut utiliser les *dialogues JavaFX.*
- Le code est propre et bien formaté
- Aucune méthode ne dépasse 30 lignes (incluant javadoc et commentaires)
- Il n'y a pas de @ID ou méthode inutile dans le code.
- Toutes les consignes ont bien été suivies et tous les comportements fonctionnent sans problème.
- Les méthodes que vous avez programmées ou modifiées ont une javadoc conforme et des commentaires pertinents.
- Les fonctions sont bien découpées. Les noms des méthodes sont clairs et significatifs. Il n'y a pas de code dupliqué ou redondant.
- Il n'y a pas de *StackTrace* dans la console *IntelliJ.*
- Des solutions efficaces ont été utilisées pour résoudre les problèmes (ex : classe *Function* sans méthode *toString*)
- La présentation de L'UI **(30%)**
- Le code de l'application **(30%)**
  - Qualité du code
    - Respect de l'architecture imposée.
    - Gestion des ressources
    - Séparation du code en méthode s

        - Commentaires (*javadoc* et commentaires)
    - Normes à respecter:
        - Nomenclature java dont:
            - Nom de méthode commence toujours par un verbe et une lettre minuscule.
        - Ne pas utiliser les instructions break ou continue (sauf break dans une switch).
        - Une seule instruction retour par méthode.
        - Les javadoc :
            - commence par un verbe;
            - tous les paramètres doivent être expliqués ainsi que les contraintes qui se rapportent à chacun d'eux.
        - Une méthode ne doit pas dépasser 30 lignes (incluant la documentation)
        - Évitez les détours inutiles et le code inutilement compliqué.
        - Utiliser des noms **significatifs complets**.
        - Un programmeur de votre niveau devrait comprendre votre code en moins de 10 secondes.
        - Utilisez des constantes ou des propriétés,
            - pas de numéros autres que 0 dans le code.
            - Pas de chaîne de textes hard codées.
- Fonctionnement **(30%)**
    - Fonctionnement de l'application principale.
    - Apparition des dialogues.
    - Comportements des équations et variables.
    - Bouton de calculatrice
    - Changement de texte sur les boutons Toggle.
    - Ajout et retrait d'équations
    - Les tests unitaires passent.
    - Absence de *StackTrace* dans la console.
    - Toutes les fonctionnalités sont bien réalisées.
- Utilisation adéquate de git (1**0%)**
    - **Le code appartient à celui qui le commit. Si vous travailler ensemble sur une portion de code comitter les 2 noms. Vous perdrez les 10% si vous ne respectez pas cette règle.**
    - **Les messages de commits doivent clair, concis et pertinents.**

À REMETTRE :

- Il est à remettre à la date indiquée sur Léa.
- Remettez votre projet complet dans une archive .**zip** sur Léa.
- Invitez le professeur à faire partie de votre équipe git.