Cours 420-203-RE ***Développement de programmes dans un environnement graphique*** Automne 2022 Cégep Limoilou Département d'informatique	Tp2 Calculatrice Événements *JavaFX* Interface *FXML* (12 %)

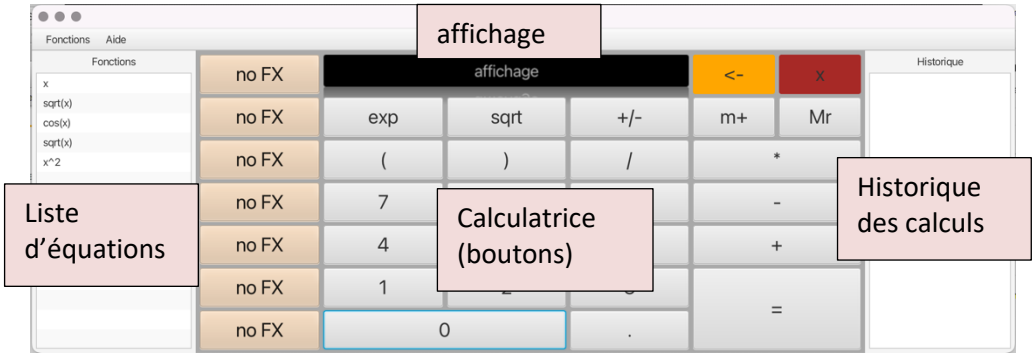
CONTEXTE DE RÉALISATION DE L'ACTIVITÉ :

- Durée : 3 semaines
- Ce travail peut être réalisé en équipe de 2 maximum
- 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.*
- Créer un code clair et concis.

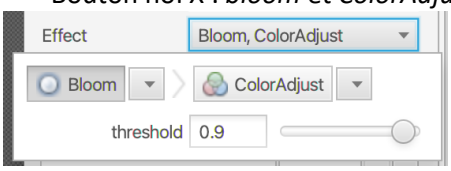
À partir du début d'interface graphique fournie, vous allez devoir réaliser la «calculatrice» suivante :



Activités à réaliser

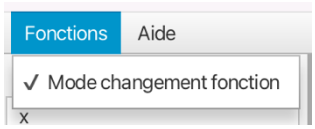
Création de l'UI :

- Vous pouvez réaliser l'UI avec *SceneBuilder* ou par programmation, mais on vous recommande fortement de le faire avec *SceneBuilder*.
    - Indices
        - La section du milieu est un *GridPane*
        - Le gestionnaire racine est un *BorderPane*
        - Tous les espaces entre les composants sont de *5px* ou *10px*
        - Styles à utiliser:
            - Couleur de fond: ***-fx-background-color***
            - Couleur de caractère : ***-fx-text-fill***
        - Effets à utiliser:
            - Bouton noFX : *bloom* et *ColorAdjust*



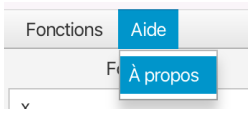
- Pour l'affichage, il y a un effet *Reflection*.

- Le vidéo ***tp2a.mov*** 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:

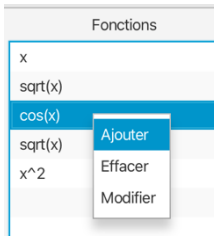


N.B. Contient une boîte à cocher

  - Aide:



- Menu contextuel dans la liste d'équations : Faites en sorte que lorsque l'utilisateur clique dans la liste de fonctions avec le bouton gauche de la souris, le menu contextuel suivant apparaisse. Il ne contient que des items normaux.

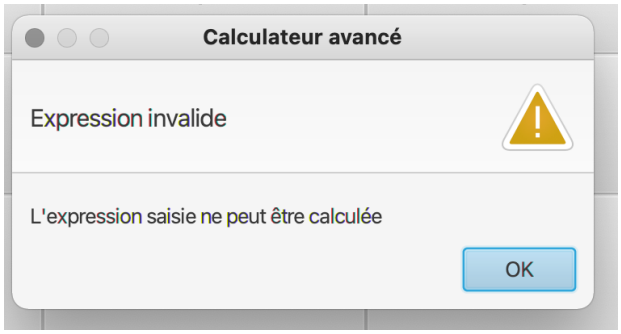


- Votre application aura à communiquer avec l'utilisateur au moyen de 2 sortes de boîtes de dialogue : la boîte standard de message/avertissement/erreur et la boîte de saisie de texte. Vous pourrez également utiliser des dialogues plus avancés à votre convenance. La classe Utilitaire ***DialoguesUtils***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

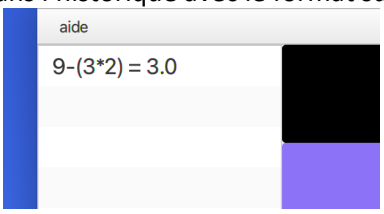
Programmation des comportements

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 transférer à un objet ***Expression*** pour que ce dernier en fasse le calcul à l'aide de sa méthode ***calculate***(). L'objet Expression est fourni par la librairie ***MathParser.org-mXparser*** qui a été préinstallée dans le projet qu'on vous a remis. Pour plus d'informations, consultez le site https://mathparser.org/. Si vous avez des problèmes avec cette librairie, consultez la section *informations utile* un peu plus loin.
  - S'il y a une erreur dans l'expression, la méthode *calculate* retournera le résultat *Double.NaN* que vous pouvez vérifier avec l'instruction ***Double.isNaN(value)***. On ne doit pas présenter ce résultat à l'utilisateur. Il faut plutôt faire sortir un dialogue qui l'informe que l'expression est invalide.

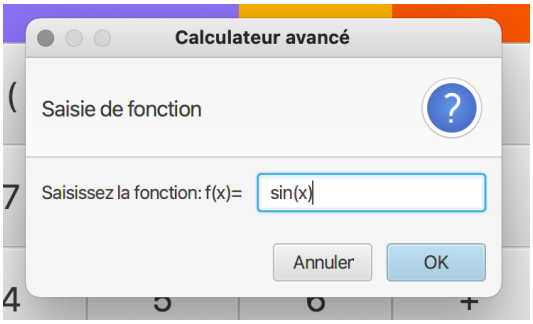


- Lorsque le calcul est valide, affichez le résultat dans l'affichage et insérez une copie de l'expression et du résultat dans l'historique avec le format suivant :

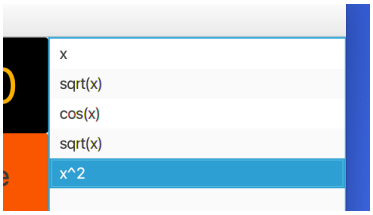


- L'expression mathématique calculée à gauche , un symbole « = » au centre, puis le résultat à droite.
- Le vidéo ***tp2-calcul.mov*** présente une opération de calcul, l'ajout dans l'historique et une expression invalide.

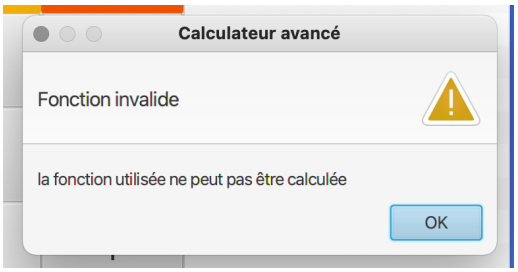
- **Boutons d'effacement** – "<-" et "*x*"
  - Lorsque l'utilisateur appuie sur le bouton d'effacement « <- » , l'application doit effacer le dernier caractère saisi par l'utilisateur. Celui qui se trouve le plus à droite de l'affichage. Faites les manipulations nécessaires sur la chaîne de caractères contenue dans l'affichage pour obtenir ce résultat.
  - Lorsque l'utilisateur appuie sur le bouton vide, l'affichage est vidé au complet.
  - Le vidéo ***tp2-effacement.mov*** présente les comportements des 2 boutons d'effacement.
- **Bouton de mémoire** – *« M + »* et *« M r »*
  - Le bouton d'ajoute en mémoire *« M+ »* permet de mettre de côté la valeur contenue dans l'affichage.
  - Le bouton de récupération de la mémoire *« Mr »* ajoute (<u>et non remplace</u>) dans l'affichage le nombre contenu en mémoire.
  - Le vidéo ***tp2-memoire.mov*** présente les comportements des 2 boutons de mémoire.
- **Liste d'équations** – à la droite de l'UI
  - Lorsqu'on appuie avec le bouton droit de la souris dans la liste d'équations, le menu contextuel que vous avez fait apparaît et offre 3 possibilités.
    - **Ajoute une équation :**
      - Les équations sont des objets de type ***Function*** fournis par la librairie *MathParser.org-mXparser*. Pour construire un objet *Function* il suffit de passer au constructeur une chaîne de caractères de la forme :
        - ***f(x)= « quelque chose avec x»***
        - Exemples : f*(x)=sin(x)*
        - Vous pouvez consulter la liste des fonctions utilisables disponibles ici : http://mathparser.org/mxparser-math-collection/
      - L'application demande à l'utilisateur de saisir une fonction/équation en utilisant un dialogue approprié. Le préfix *f(x)* ne doit pas être saisi par l'utilisateur.



- Notez bien que l'objet *Function* ne possède pas de méthode *toString* adéquate. Vous devrez donc trouver un moyen pour que l'équation s'affiche correctement dans la liste d'équations.



- Avant d'ajouter une équation dans la liste, il faut s'assurer que cette équation est valide en l'essayant. Pour faire calculer une fonction, il faut appeler sa méthode ***calculate(double)*** . Par exemple
  - *aFunction.calculate(5.3)*
- Si la fonction saisie par l'utilisateur n'est pas valide, il faut l'avertir et ne pas ajouter la fonction invalide dans la liste.



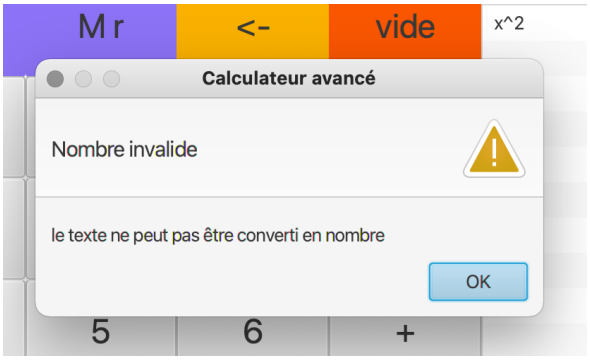
- **Efface l'équation sélectionnée**
  - Ce menu efface simplement l'équation sélectionnée de la liste d'équations. Si aucune équation n'est sélectionnée, ce menu ne doit rien faire. Il ne doit pas y avoir de *StackTrace* dans la console *IntelliJ*.
- **Modifie l'équation sélectionnée**
  - Effectue la même chose que *ajoute une équation,* mais au lieu d'ajouter l'équation elle modifie celle qui est déjà sélectionnée.
  - Si aucune équation n'est sélectionnée, il ne doit rien se passer (pas de *StackTrace*)
  - Il est important que le dialogue qui demande la modification présente à l'ouverture le texte de la fonction à modifier.

  - Tous les comportements du menu contextuel de la liste d'équations sont présentés dans le vidéo ***tp2-liste-equations.mov***.

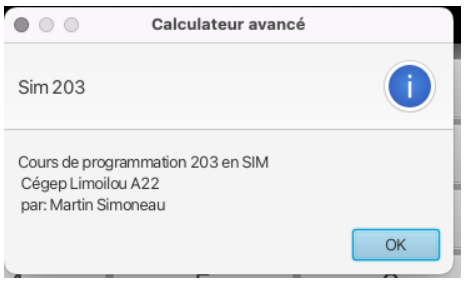
- Double clic dans les listes (historique et liste d'équations).
  Ces fonctionnalités doivent être ***complètement réalisées par programmation*** et non avec *SceneBuidler*. Vous pourrez brancher les comportements dans la méthode *initialise*() de la classe *CalculatriceController*.
  - Lorsque l'utilisateur double-clique sur une entrée de l'historique, la valeur contenue dans cette dernière doit être ajoutée à l'affichage .

- Lorsque l'utilisateur double-clique sur la liste d'équations. L'équation sélectionnée doit être calculée avec la valeur contenue dans l'affichage.
  - Cette fonctionnalité commence par récupérer le contenu de l'affichage dans une variable de type *double*. Elle récupère ensuite l'équation sélectionnée de la liste d'équations. Elle lance le calcul de la fonction pour la valeur contenue dans l'affichage. Si le résultat n'est pas *NaN*, elle remplace le contenu de l'affichage avec le résultat obtenu.
    - Lorsque le calcul est effectué. Le résultat doit être placé dans l'historique en respectant le format *expression = résultat.* Par exemple :

      cos(15)= -0.7596879
  - En aucun cas, il ne doit y avoir de *StackTrace* dans la console *IntelliJ*.
  - Si le texte contenu dans l'affichage n'est pas un nombre valide, l'application doit afficher le message suivant :



- **Menu**
  - Si le menu *Fonctions/Mode changement de fonction* est activé,
    - en appuyant sur un bouton de fonction (les *no FX*) le programme va assigner la fonction sélectionnée dans la section gauche à ce dernier s'il y a une fonction de sélectionnée.
  - Si le menu *Fonctions/Mode changement* n'est pas activé,
    - la fonction emmagasinée sur le bouton sera calculée sur l'affichage, avec la valeur contenue dans l'affichage comme valeur de x.
  - 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 :
    - *f(x)=x*
    - *f(x)=sqrt(x)*
    - *f(x)=sin(x)*
    - *f(x)=cos(x)*
    - *f(x)=x^2*
- **Animation des touches de la calculatrice (bonus de 5 %)**

Cette fonctionnalité doit être ***complètement réalisée par programmation*** et non avec *SceneBuidler*. Vous pourrez brancher les comportements dans la méthode *initialise*() de la classe *CalculatriceController*.

- Lorsque la souris clique sur un bouton:
  - Le bouton rapetisse à 80% avec le bouton qui s'enfonce et retourne à 100% lorsque le bouton de la souris est relâché ou lorsque la souris quitte le bouton avant le relâchement.
  - Évitez de placer un callback sur chaque bouton!
  - Indice:
    - utilisez la méthode ***setScale****()* pour y parvenir.
    - Attention! le *clic* du bouton est un *handler* qui consomme l'événement.
- Lorsque l'utilisateur appuie sur l'une des touches (0,1,2,3,4,5,6,7,8,9,+,-,*,/,=,"(",")") :
  - La touche doit s'enfoncer comme si l'utilisateur avait cliqué dessus. La bonne fonctionnalité doit être déclenchée. La touche *backspace* efface le dernier caractère et la touche *delete* vide l'affichage.
  - Indices:
    - k*eyboardEvent.getText()* retourne le texte de la touche enfoncée.
    - *keyboardEvent.getCode()* retourne le code de la touche enfoncée.

Informations utiles

- ListView
  - Pour consulter les éléments d'un ListView
    - *List.getItems()*
  - Pour connaître l'élément sélectionné
    - *Liste.getSelectionModel().getSelectedItem();*

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 50 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*)

À REMETTRE :

- Il est à remettre à la date indiquée sur Léa.
- Remettez votre projet complet dans une archive .**zip** sur Léa.