Cours 420-203-RE ***Développement de programmes dans un environnement graphique*** Automne 2023 Cégep Limoilou Département d'informatique	Tp1 Première programme (8 %) • **JavaFX**

OBJECTIFS

- Analyser et reproduire une interface graphique.
- Utiliser les composants *JavaFX* pour créer une interface graphique;
- Comprendre et utiliser les gestionnaires de disposition.
- Programmer le comportement d'un bouton.

ACTIVITÉS À RÉALISER

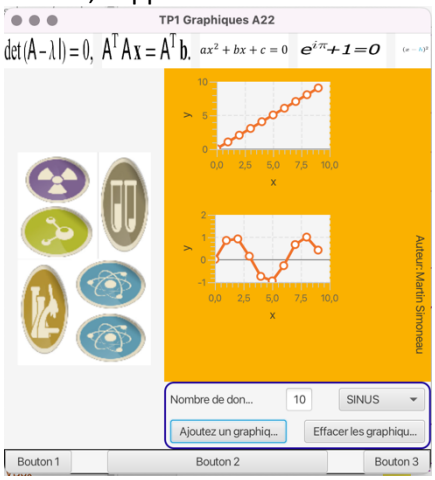
Vous avez à concevoir une interface graphique en respectant l'image du prototype fourni.

DURÉE DE L'ACTIVITÉ :

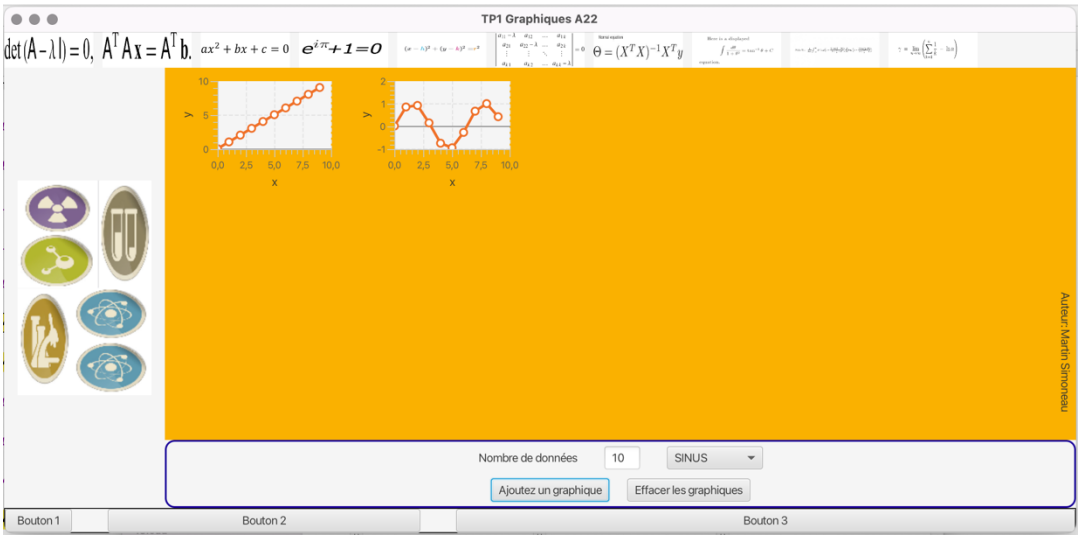
- ~2 semaines
- Ce travail peut être réalisé en équipe de 2ou 3 maximum.

PROTOTYPE 1 :

- Lorsque la fenêtre est à sa taille minimale, l'application ressemble à :



- Lorsque la fenêtre est à sa largeur maximale (la hauteur n'est pas limitée), l'application ressemble à :



Notez que :

- Vous avez reçu une vidéo nommée ***tp1-a.mov***, qui vous montre le comportement de la fenêtre lorsqu'on modifie sa taille.
- La fenêtre n'a pas de hauteur maximale, mais elle a une largeur maximale de **1200px**

ÉTAPE 1, RÉFLEXION SUR LA CONCEPTION GRAPHIQUE

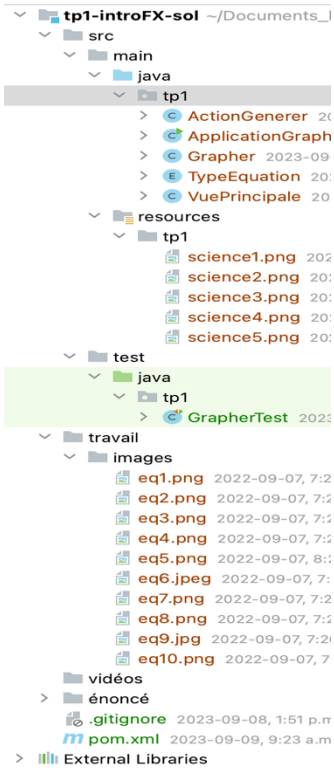
Votre première tâche consiste à identifier sur le prototype d'interface graphique présentée ci-dessus chacune des parties de vos interfaces, les gestionnaires de disposition proposés et les composants requis. Mettre vos réponses dans un document comprenant des textes d'identification des parties et les dessins pour illustrer votre réflexion (illustrer les interfaces). Voici un exemple de ce que j'aimerais:

- Faites valider votre croquis avant de commencer la programmation.
- Vous pouvez aussi faire le croquis à la main et le prendre en photo.

Informations utiles

- Dans cette scène, on retrouve au moins
  - 1 GridPane (pour les images à gauche)
  - 1 *TilePane*
  - *1 AnchorPane*
  - *1 StackPane*
  - Plusieurs *VBox* et *HBox*
  - 1 *BorderPane*

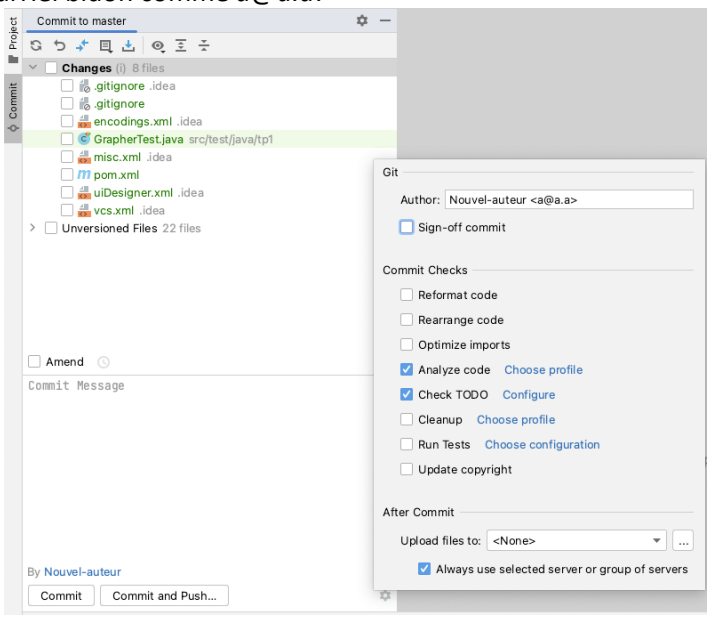
ÉTAPE 2 : PROGRAMMATION DES INTERFACES GRAPHIQUES



Votre interface doit être codée en *JavaFX* directement. Disposez votre code en *packages* et vos ressources dans un dossier de ressources, telle que présentée précédemment. ApplicationGraph est très légère, elle ne fait que présenter la scène que lui fournit la classe *VueForme*. La classe « ***VuePrincipale*** » prépare l'ensemble de l'interface, gardez le code comme présenté dans les exemples.

Informations utiles

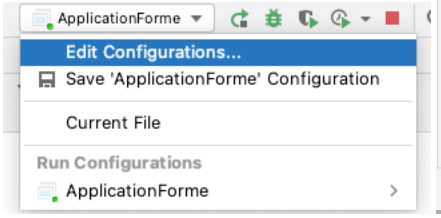
- Pour changer le nom de l'auteur au moment du commit: le nom doit avoir la forme ***noms <mail>*** . Vous pouvez entrer un courriel bidon comme *a@a.a*.

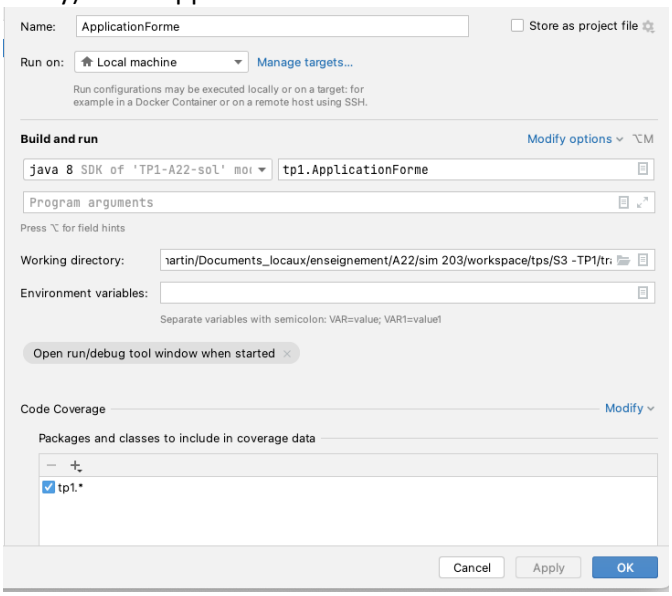


- Dimensions
  - Estimez la taille minimale de la fenêtre
  - La largeur maximale de la fenêtre est de 1200 px.
  - Les graphiques de sciences ont :

        - Largeur max de 200
        - Hauteur max de 150
    - Les images de science ont uniquement 100px ou 200px comme largeur (référez-vous à l'image)
    - La taille des 3 boutons change avec la taille de la fenêtre, observez attentivement leur comportement.
- Pour vous aider à dimensionner les éléments graphiques, on a laissé plusieurs constantes provenant de la solution dans la classe *VuePrincipale*.
- Pour faire un calcul avec MathXParser
    - `Function function = new Function("f(x)=sin(x)");`
      Ici, la partie " f(x)= " est obligatoire pour définir une fonction
    - Pour utiliser la fonction:
      `Double reponse = function.calculate(i)`
    - Si la fonction n'arrive pas à s'exécuter, elle retournera la valeur Nan qu'on peut détecter avec:
      `Double.isNaN(reponse);`
      Ne pas faire *reponse==Double.NaN*… ça ne fonctionne pas!

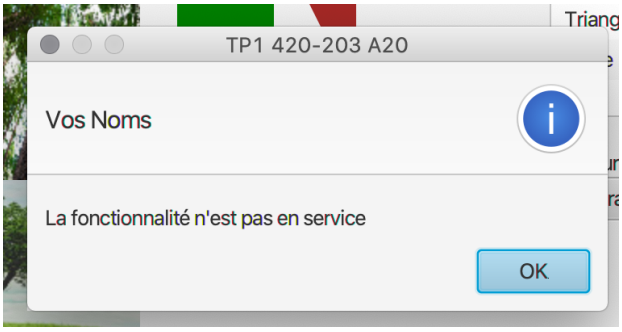
- Fichier:
    - Objet qui permet de rechercher tous les fichiers dans un dossier:
      ```
      File dossier = new File("./path");//relatif au working
      dossier.listFiles()
      ```
    - Pour créer une iage à partir d'un objet File:
      `new Image(file.toURL().openStream());`
- Pour régler le dossier de travail (working directory) d'une application dans IntelliJ:





Exigences

- Votre application doit reproduire le plus fidèlement possible les comportements de la fenêtre lorsqu'on la redimensionne (taille des boutons, positionnement des éléments…). Consultez le vidéo *tp1a.mov* pour les détails.
- Toute la fabrication de l'interface graphique doit être dans la classe *VuePrincipale*.
- L'application *ApplicationGraph* utilise la classe *VuePrincipale* pour créer la scène et elle règle les dimensions de la fenêtre.
- Dans le haut de chacune des classes, vous devez mettre les noms des auteurs.
- Lorsqu'on appuie sur n'importe quel bouton du bas (*Bouton1*, *Bouton2* et *Bouton3*), le dialogue suivant apparaît.



- Dans le nom du projet (ar*tifact ID* avec Maven), vous devez restructurer le nom *tp1-introFX-etu* par ***tp1-introFX-vos-initiales*** en remplaçant ***vos-initiales*** par vos propres initiales.
- Découpez vos classes en plusieurs méthodes de façon adéquate. Évitez les méthodes de plus de 30 lignes (incluant la documentation et les commentaires) , c'est difficile à maintenir.
  La barre du haut doit présenter toutes les équations du dossier *travail/images* peu importe leur nombre. Ce dossier ne doit pas être dans les ressources du programme, car son contenu peut être changé par l'utilisateur.
- La classe ***Grapher*** sert à créer un graphique respectant les paramètres reçus de l'interface. Les paramètres sont transmis dans un objet de type ***Grapher.Parameters*** qui contient les informations pertinentes (cette classe interne vous est déjà fournie). Les seuls objets *JavaFX* que devrait voir *Grapher* sont les classes liées au *LineChart*.
- Faites en sorte que lorsqu'on appuie sur le bouton *Ajouter un graphique* :
  - *VuePrincipale* envoie les informations nécessaires (type de graphique, nombre de points et nom de la courbe) à la classe *Grapher* pour que ce dernier puisse fabriquer le *LineChart* correspondant. La classe *Grapher* utilise la classe *Function* de *MathParser* pour calculer le résultat:
    - La classe *Grapher* reçoit un objet de type *Grapher.Parameters* (une classe interne).
    - On crée on objet de type ***Function*** comme suit:
      ```
      Function fx = new Function("f(x)=x^2");
      ```
    - On fait ensuite le calcule pour une valeur x:
      ```
      fx.calcule(valX);
      ```
    - Elle doit produire une courbe *XYChart.Series* avec le bon nombre de données, le bon titre et le bon type de courbe.
      - SINUS: courbe sin(x)
      - LINEAR: x
      - QUADRATIC: x^2
      - INVERSE: 1/x. (attention à ne pas mettre de valeur infinie ou NaN dans le graphique)
    - Les données en x partent toujours de 0 et font des incréments de 1.
  - Vous pouvez utiliser la classe ***ActionGenerer*** ou utiliser une autre méthode de *callback* vue en classe.
  - Le UI doit ajouter le *LineChart* dans la section centrale de votre application. Chaque fois qu'on appuie sur le bouton, un nouveau *LineChart* est créé avec les dernières données saisies.
- Faites en sorte que lorsqu'on appuie sur le bouton *Effacer les graphiques* :
  - Tous les graphiques de la section centrale sont effacés.
- Mettez des commentaires appropriés dans le code et faites la *javadoc*.
- Le vidéo ***tp1-b.mov*** montre le comportement de l'application lorsqu'on appuie sur le bouton *Générer*.
- Les noms des auteurs dans la partie centrale:
  - Ils sont superposés au-dessus de la zone en orange
  - Ils sont toujours en bas et à gauche de cette zone
  - Pour les faire tourner et les positionner correctement, il faudra utiliser les méthodes:
    -

- **Barème D'évaluation:**

- La présentation de L'UI **(30%)**
    - Conformité statique (15%)
        - Espacements
        - Marges
        - Choix des composants
        - fidélité
    - Gestion du redimensionnement (15%)
- Le code de l'application **(30%)**
    - Code de fabrication des graphiques
        - *Grapher*
        - Action du bouton *générer*
    - Qualité du code
        - Respect de l'architecture imposée.
        - Gestion des ressources
        - Séparation du code en méthode
        - Méthodes bien découpées avec un maximum de 50 lignes.
        - 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.
        - Un programmeur de votre niveau devrait comprendre votre code en moins de 10 secondes.
        - Utilisez des constantes ou des propriétés,
            - pas de numéros autre que 0 dans le code.
            - Pas de chaîne de textes hard codées.
- Fonctionnement **(30%)**
    - Fonctionnement de l'application principale
    - Apparition des dialogues.
    - Création des *LineChart* qui respectent les paramètres de l'UI
    - Absence de *StackTrace* dans la console.
- 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.**

À REMETTRE :

- Remettez votre projet complet, incluant votre croquis, dans une archive .**zip** sur Léa à La date indiquée sur Omnivox/Léa.