Développement de programme dans un environnement graphique 420-203-LI Automne 2022 Cégep Limoilou Département d'informatique Professeur : Martin Simoneau	**Formatif 6** *Services périodiques* *Service et affichage*

Objectifs

- Modifier l'UI avec un service
- Utiliser des services périodiques

À remettre :

- Le travail sera remis sur Léa à la date indiquée.

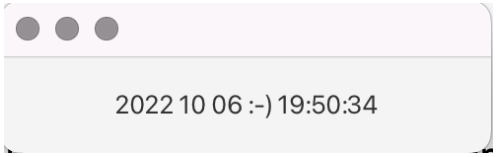
Contexte :

- Remettre vos projets sur Léa
- Une remise par étudiant

À faire

- Ouvrez le projet ***formatif6-etu***

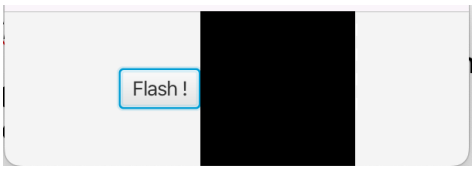
1. Faire une horloge



a. Dans le sous-package *clock*, ouvrez la classe *ClockService*
   i. Le service crée une boucle qui continue jusqu'à ce qu'elle soit interrompue. Chaque seconde, la boucle retourne l'heure et la date actuelle. Pour vous aider:
      1. L'interruption du thread peut être testée avec
         a. *Thread.interrupted()*
      2. *SlowTimehelper* peut vous aider à attendre une seconde
      3. *LocalDateTime.now()* fournit l'heure et la date à l'instant de son appel.
      4. La méthode *updateValue* peut vous permettre de retourner chaque valeur.

b. Dans la classe *ClockApplication*
   i. Utilisez le service *ClockService* pour faire afficher l'heure et la date dans le Label *root*. L'horloge est lancée dès l'ouverture de l'application. Pour vous aider:
      1. *DateTimeFormatter.ofPattern()* vous permet de créer un objet *DateTimeFormatter* qui peut être transmis à la méthode *format* d'un objet *LocalDateTime*.
      2. Pour consulter la dernière valeur, branchez un *listener* sur la propriété *lastValue* du service
      3.

2. Faire scintiller un nœud



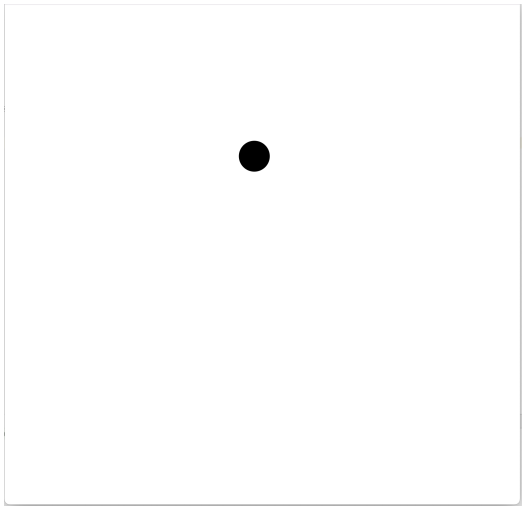
a. Dans le sous-package *flasher*, ouvrez la classe ***FlasherAnimationService***
   i. Faites de cette classe un service avec répétition (*ScheduledService*) qui retourne un boolean
   ii. À chaque nouvel appel de call, le service doit simplement alterner la valeur de retour entre vrai et faux.

b. Dans la classe **FlasherApplication**
    i. Créez une instance de FlasherService, et programmez-le pour:
        1. Avoir une période de 0.22 seconde
        2. Démarrer lorsque le bouton de la souris est tenu sur le bouton "Flash!"
        3. Arrêter lorsque le bouton est relâché
        4. Lorsque la valeur du service *lastValue* change, il faut faire alterner la couleur du rectangle entre blanc et noir. Pour vous aider:
            a. *setFill(Paint p)* permet de changer la couleur d'un nœud.
            b. Assurez-vous que la valeur de *lastValue* n'est pas nulle.

3. Faire déplacer une fenêtre

a. Dans le sous-package *uiAnimation*, ouvrez la classe ***WindowAnimationService***
    i. Faites de cette classe un service avec période qui retourne un objet de type *LocationTaille* (classe interne) à chaque appel.
    ii. Pour utiliser ce service, il faut lui transmettre deux objets de type *WindowAnimationService.LocationTaille* au moyen d'un setter adéquat.
        1. *L'un correspond à la location et à la taille souhaitée*
        2. *L'autre correspond à la location et à la taille actuelle de la fenêtre.*
    iii. À chaque appel, le service doit calculer une nouvelle valeur actuelle en se rapprochant de la valeur souhaitée pour chaque valeur (x, y, largeur et longueur) d'un incrément correspondant à *locationIncrement* (pour x et y) et *tailleIncrement* (pour longueur et largeur). Pour vous aider:
        1. Attention le service *JavaFX* fait de la coalescence lorsque l'objet reçu ne varie pas (basé sur la référence et non sur les valeurs).
        2. Si la distance à parcourir est moindre que l'écart typique, il ne faut pas dépasser
        3. L'animation doit s'arrêter lorsque les valeurs actuelles ont atteint toutes les valeurs souhaitées.
b. Dans la classe ***UiAnimationApplication***
    i. Instanciez un service WindowAnimationService
    ii. Régler la période à 0.03 seconde
    iii. Lorsque la valeur de *lastValue* change, modifier la taille et la position de la fenêtre en fonction de *LocationTaille* reçue (attention aux valeurs nulles)
    iv. Lorsqu'on appuie sur le bouton de la souris
        1. Instanciez un objet *LocationTaille* basée sur les valeurs actuelles du primaryStage et transmettez-le au service comme valeur actuelle
        2. *Instanciez* un objet *LocationTaille* basé sur les valeurs actuelles du *primaryStage* mais avec un décalage correspondant aux différents *TextView*. Transmettez-le au service comme valeur souhaitée
        3. Redémarrez le service à chaque fois qu'on appuie sur le bouton.

1. Simuler un projectile



c. Dans le sous-package ***projectile***, ouvrez la classe *ProjectileAnimationService*:
    i. On vous fournit la classe interne *Etat*. Elle sert à contenir tous les états variables de la simulation. Notez que cette classe est *Clonable*, on peut donc cloner une instance pour et obtenir une nouvelle instance qui contient les mêmes valeurs d'attribut.
        1. Les vitesses vx et vy
        2. Les position x et y
        3. Les accélérations ax et ay
        4. La masse m (constante de 1 kg)
        5. La force en x fx (=0)
        6. La force en y fy (=-9,8)
    ii. Votre service doit d'abord recevoir un état initial par le constructeur.
    iii. La tâche de ce service va devoir calculer chacun des états à chaque pas de temps, mais il faut d'abord déterminer exactement le temps qui s'est écoulé depuis le dernier pas de temps. Pour y arriver
        1. Utiliser la méthode *System.currentTimeMillis()* qui retourne un *long* correspondant au nombre de millisecondes qui se sont écoulées depuis un temps de référence. Il importe donc de prendre une lecture en début de simulation pour pouvoir calculer le temps qui s'est écoulé au premier pas de temps. Ensuite, l'incrément *dt* sera la différence de millisecondes entre 2 pas de temps consécutifs. Attention, il faut convertir le temps en seconde!
        2. Une fois l'intervalle de temps connu, vous devez appeler une méthode qui va modifier les états en fonction des équations physiques du mouvement.
            a. ay = fy / m
            b. vy = vy + ay ***dt**
            c. y = ½ ax **dt** + vy **dt+ y**
            remarquez que y dépend du y précédent et vy du vy précédent . Il va falloir être méticuleux pour ne pas mélanger les nouveaux états calculés avec les anciens.
        3. La tâche tourne en boucle jusqu'à ce qu'elle soit annulée ou lorsque la position y du projectile sort du cadre. On définit le cadre de la simulation allant de 0 jusqu'à 1000 en largeur et de 0 jusqu'à 800 en hauteur. Chaque pixel correspond donc à 1 mètre.
        4. Une fois le nouvel état calculé, il faut le retourner au service à chaque pas de temps
        5. N'oubliez pas de ralentir la boucle principale de 15ms pour ne pas surchauffer l'ordinateur.
    iv. Allez dans la classe *ProjectileApplication*
        1. Cette simulation sera lancée directement au lancement de l'application
        2. Vous devez d'abord créer un état initial
            a. x = 15 s
            b. y = 15 s
            c. vx = 15 m/s

   d. vy = 50 m/s
   e. ax = 0 m/s2
   f. ay = -9,8 m/s2 (ou N/KG)
   g. m = 1 kg
3. On doit ensuite programmer le service pour afficher chacun des états intermédiaires au fur et à mesure que la simulation avance. Attention les coordonnées en y de *JavaFX* sont inversées. Pour faire la transposition il faut donc connaître la taille de la fenêtre (1000 de hauteur).
4. Démarrer votre service et déboguer le tout jusqu'à ce que ça fonctionne.

FIN