



**ENSTA
BRETAGNE**

Java object-oriented programming

Énoncés des Travaux Pratiques

Hiba Hnaini
Sylvain Guérin

Année scolaire 2022-2023

Contents

Énoncés des travaux pratiques	2
Présentation du Document	2
Avant de commencer à développer les exercices.....	6
Comment créer un nouveau projet dans IntelliJ	6
GitHub	15
1 Notions de base du langage et fonctions anonymes	22
1.1 No Pain, No Gain	22
1.2 Un peu de récursivité.....	22
1.3 Méthodes de tri	23
1.3.1 Tri bulle (Bubble Sort)	23
1.3.2 Tri par insertion (Sort by Insertion).....	24
1.4 Température par ville	25
2 Notions de base de la programmation orientée objet et les génériques de Java	26
2.1 Dépôt de voitures d'occasion (@The Australian National University)	26
2.2 Compte bancaire.....	26
2.3 Taxes	28
2.4 Dossiers des étudiants	30
2.5 Cercles.....	30
3 Notions avancées de la POO	33
3.1 Bibliothèque (inspiré des TPs de Laurent Tichit)	33
3.2 Les polygones.....	36
3.3 Simulation d'un mini-écosystème de cigales et fourmis	38
3.4 Expressions arithmétiques (inspiré des TPs de Laurent Tichit)	42
3.5 Développer une hiérarchie des comptes créditeurs	45
4 Gestion d'erreurs	45
4.1 Exceptions (© Dr Robert Harle).....	45
4.2 Lecture d'un fichier	46
4.3 Les numéros (© Chua Hock-Chuan)	46
4.4 Compte bancaire.....	48
4.5 Factorials.....	48
5 JavaFX	51
5.1 Interface Graphique Simple	51
5.2 Simple Transitions (The Australian National University)	51
5.3 Voitures de course.....	51
5.4 Menus (PARIS DIDEROT)	51
5.5 SliderDemo	53

Énoncés des travaux pratiques

Présentation du Document

Ce document a été conçu afin de vous aider à atteindre, par la pratique, les objectifs du cours. Les objectifs du cours ont un rapport direct avec le contenu du cours et ce document vous aidera à mettre en pratique chaque concept abordé dans ce plan de travail, pendant les séances de TPs :

- 1. Introduction et Notions de base du langage (2h cours + 2h TP) :** propriétés de Java, Java vs Python, classes, attributs, types primitifs, tableaux, méthodes, input scanner, lecture/écriture de fichiers, structures de contrôle, instructions itératives et de sélection, méthode incrémentale de conception, au-delà des bases pour écrire un bon code Java.
- 2. Notions de base de la POO (2h cours + 2h TP) :** objets, classes, attributs, méthodes (constructeurs, getters/setters, logique métier), instanciation, surcharge, protection, variables et méthodes de classe vs variables et méthodes d'instance.
- 3. Notions avancées de la POO (2h cours + 2h TP) :** héritage, polymorphisme, typage statique (opérateur *cast*), résolution de variables, résolution de méthodes, redéfinition de méthodes, classes et méthodes abstraites, interfaces.
- 4. Gestion des erreurs (2h cours + 2h TP) :** importance de la qualité des logiciels, *Java errors*, Exceptions, try-with-resources, éviter les try-catch?, bugs, JUnit et les test unitaires.
- 5. Interface homme-machine (2h cours + 2h TP) :** installation et introduction à JavaFx, anatomie d'une application JavaFx, éléments graphiques, définir le comportement, layout, transformations, Charts, HTML content, Media, transitions animées, pièges à éviter.

Comment utiliser ce document ?

Sélectionnez les exercices qui vous intéressent le plus et essayez de faire ce qu'on demande pour chaque exercice. Si votre programme ne fonctionne pas du premier coup, rappelez-vous de la consigne de Tomas Edison "Genius is one percent inspiration and ninety-nine percent perspiration". Réessayez alors avec persévérance. Veuillez utiliser les séances de TP pour poser les questions sur les points qui vous auront posé un problème, nous travaillerons alors sur ces points.

Le document comporte des exercices divers et variés sur tous les sujets du cours ; vous n'avez donc pas d'excuse pour ne pas pratiquer les concepts vus dans les cours magistraux.

Programme du cours	Contenu/Compétences	TPs
Introduction et Notions de base du langage (2h cours + 2h TP)	Propriétés de Java	No Pain, No Gain ; Un peu de récursivité
	Classes	No Pain, No Gain ; Un peu de récursivité
	Attributs	Un peu de récursivité

	Types primitifs	No Pain, No Gain ; Un peu de récursivité ; Méthodes de tri
	Tableaux	Un peu de récursivité ; Méthodes de tri
	Méthodes	Un peu de récursivité ; Méthodes de tri
	Input Scanner	Température par ville
	Lecture/écriture de fichiers	Température par ville
	Structures de contrôle	Méthodes de tri ; Température par ville
	Instructions itératives et de sélection	Méthodes de tri ; Température par ville
	Méthode incrémentale de conception	Un peu de récursivité ; Méthodes de tri
Notions de base de la POO (2h cours + 2h TP)	Objets	Dépôt de voitures d'occasion ; Compte bancaire ; Taxes ; Dossiers des étudiants ; Cercles
	Classes	Dépôt de voitures d'occasion ; Compte bancaire ; Taxes ; Dossiers des étudiants ; Cercles
	Attributs	Dépôt de voitures d'occasion ; Compte bancaire ; Taxes ; Dossiers des étudiants ; Cercles
	Méthodes	Dépôt de voitures d'occasion ; Compte bancaire ; Taxes ; Dossiers des étudiants ; Cercles
	(Constructeurs, getters/setters, logique métier)	Dépôt de voitures d'occasion ; Compte bancaire ; Taxes ; Dossiers des étudiants ; Cercles
	Instanciation	Dépôt de voitures d'occasion ; Taxes ; Dossiers des étudiants ; Cercles

	Surcharge	Voir tous les exercices de la section Notions avancées de la POO
	Protection	Dépôt de voitures d'occasion ; Compte bancaire ; Taxes ; Dossiers des étudiants ; Cercles
	Variables et méthodes de classe vs variable et méthodes d'instance	Compte bancaire ; Dossiers des étudiants ; Cercles
Notions avancées de la POO (2h cours + 2h TP)	Héritage	Bibliothèque ; Polygones ; Simulation d'un mini-écosystème de cigales et fourmis ; Expressions arithmétiques ; Développer une hiérarchie des comptes créditeurs
	Polymorphisme	Bibliothèque ; Polygones ; Simulation d'un mini-écosystème de cigales et fourmis ; Expressions arithmétiques ; Développer une hiérarchie des comptes créditeurs
	Typage	Bibliothèque ; Polygones ; Simulation d'un mini-écosystème de cigales et fourmis ; Expressions arithmétiques
	Statique (opérateur cast)	Bibliothèque ; Polygones
	Résolution de variables	Polygones ; Simulation d'un mini-écosystème de cigales et fourmis ; Expressions arithmétiques
	Résolution de méthodes	Bibliothèque ; Polygones ; Simulation d'un mini-écosystème de cigales et fourmis ; Expressions arithmétiques ; Développer une

		hiérarchie des comptes créditeurs
	Redéfinition de méthodes	Bibliothèque ; Polygones ; Simulation d'un mini-écosystème de cigales et fourmis ; Expressions arithmétiques ; Développer une hiérarchie des comptes créditeurs
	Classes et méthodes abstraites	Bibliothèque ; Expressions arithmétiques ; Développer une hiérarchie des comptes créditeurs
	Interfaces	Expressions arithmétiques ; Développer une hiérarchie des comptes créditeurs
Gestion des erreurs (2h cours + 2h TP)	Exceptions	Exceptions ; Lecture d'un fichier ; Les numéros (© Chua Hock-Chuan) ; Factorials
	try-with-resources	Exceptions ; Lecture d'un fichier ; Factorials
	JUnit et les test unitaires	Les numéros (© Chua Hock-Chuan) ; Compte Bancaire
Interface homme-machine (2h cours + 2h TP)	Éléments graphiques	Interface Graphique Simple ; Simple Transitions ; Voitures de course ; Menus ; SliderDemo
	Définir le comportement	Interface Graphique Simple ; Simple Transitions ; Voitures de course ; Menus ; SliderDemo
	Layout	Simple Transitions ; Voitures de course ; Menus ; SliderDemo

	Transformations	Simple Transitions ; Voitures de course ; SliderDemo
	Charts, HTML content, Media	----
	Transitions animées	Simple Transitions ; Voitures de course

Avant de commencer à développer les exercices

Veillez vérifier que vous avez un IDE appropriée pour le développement en Java. En étant étudiant ou salarié de l'école, vous pouvez installer IntelliJ Ultimate (version pro).

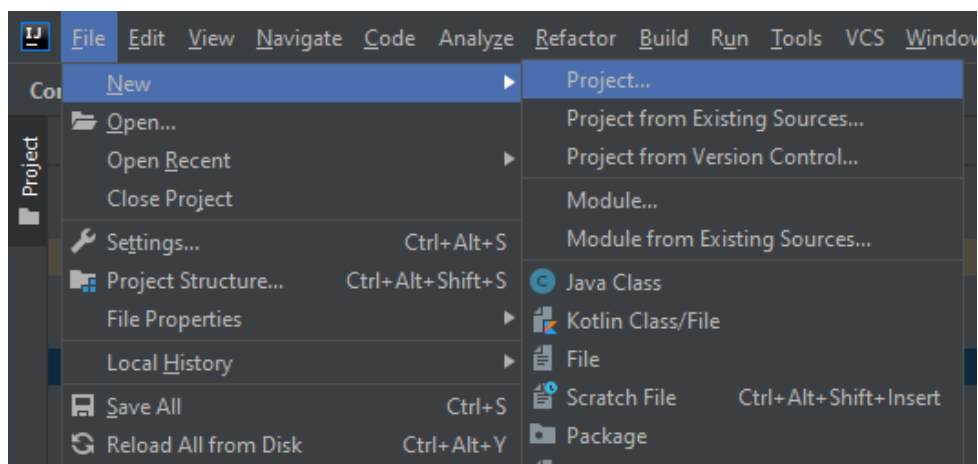
Pour installer IntelliJ Ultimate il faut obtenir une licence pro du JetBrains Product Pack for Students.

Afin de demander une licence individuelle pour une utilisation personnelle 'éducation', les étudiants et les enseignants doivent utiliser leurs adresses mail de l'école (prenom.nom@ensta-bretagne.org pour les étudiants et prenom.nom@ensta-bretagne.fr pour le personnel) pour formuler leurs demandes de licence en utilisant [ce formulaire](#).

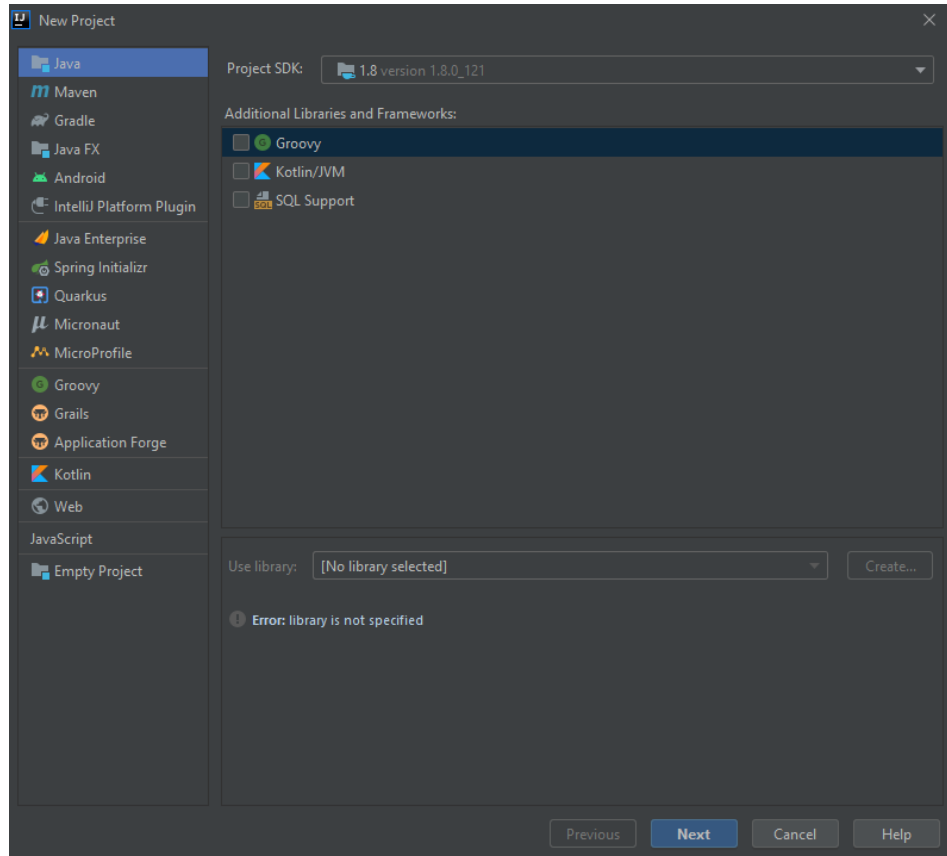
Après validation de votre formulaire, un mail de confirmation vous sera envoyé nécessitant une validation de votre part en cliquant le lien fourni.

- Confirmer votre inscription via le lien donné dans le mail 'Confirm Request' pour accéder à la page de connexion/création d'un compte JetBrains.
- Connectez-vous avec votre compte ou créer en un nouveau en utilisant toujours votre adresse mail de l'école
- Une fois connecté, vous pouvez télécharger la dernière version de [IntelliJ IDEA Ultimate](#) et d'autres produits recensés dans la liste 'Downloads'. Installer l'application IntelliJ IDEA que vous venez de télécharger, et dans le processus d'installation veuillez fournir votre email de l'école et le mot de passe que vous avez choisi lors de la création de votre compte JetBrains.

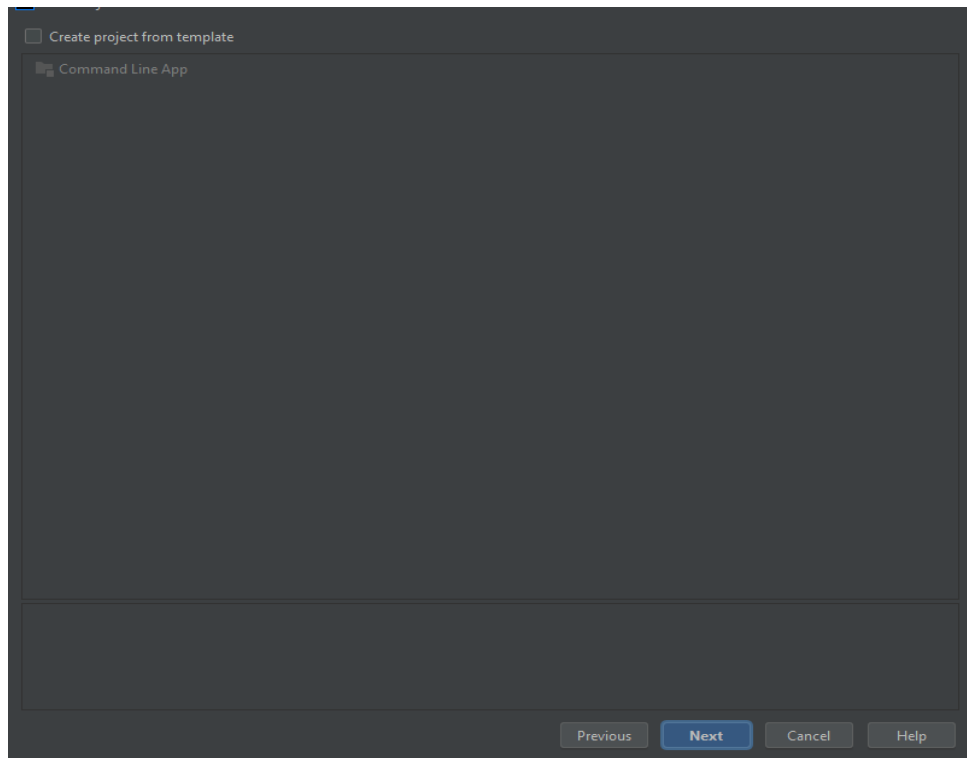
Comment créer un nouveau projet dans IntelliJ



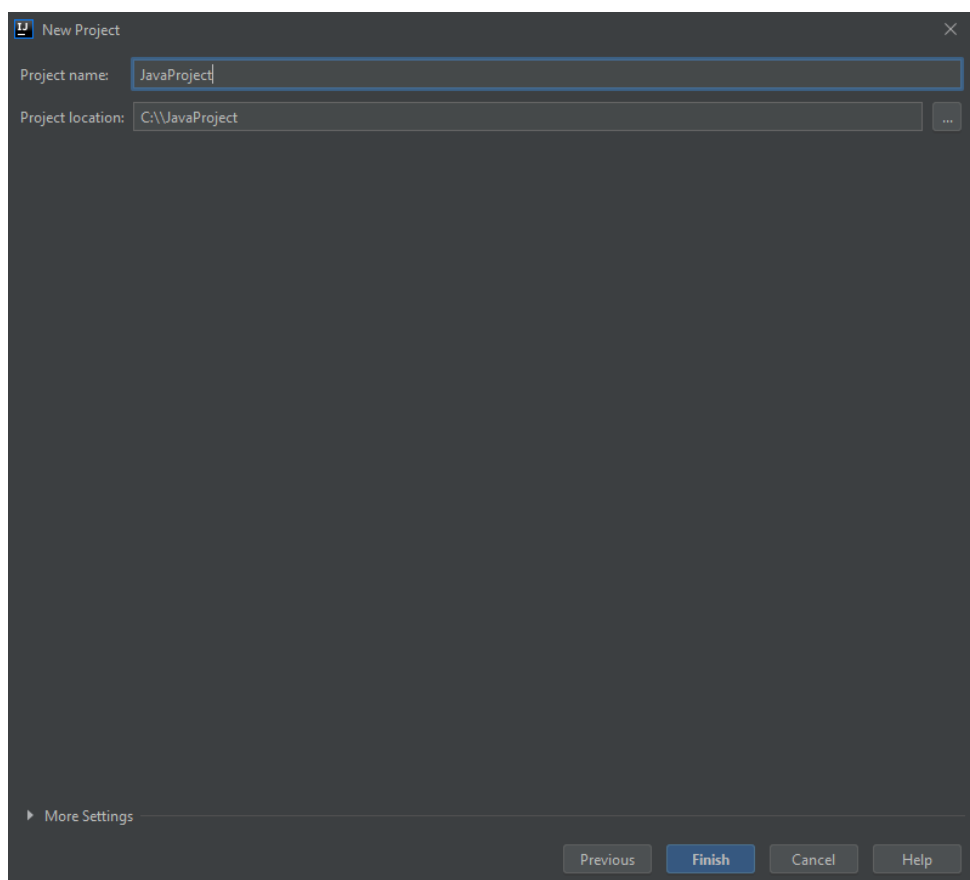
- 1) Pour créer un nouveau projet dans IntelliJ, naviguez vers Fichier->Nouveau->Projet ou File->New->Project dans la version anglaise.
- 2) La fenêtre représentée dans l'image suivante devrait apparaître. Choisissez Suivant «Next » si le type de projet, le sdk et les bibliothèques sont correctement choisis. Dans notre cas, le type de projet est Java, le SDK java devrait être automatiquement choisi à partir de l'installation java sur votre ordinateur, et aucune bibliothèque supplémentaire n'est nécessaire.

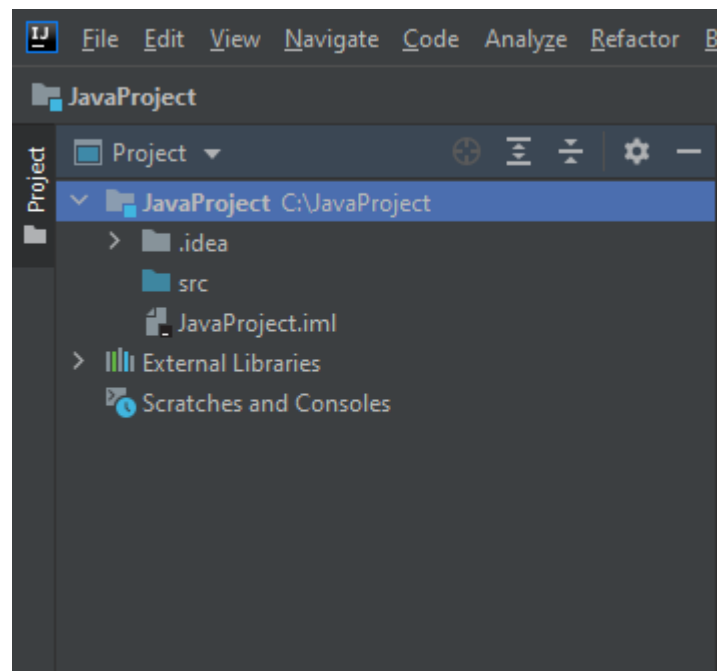


- 3) Ensuite, si vous ne créez pas un projet à partir d'un modèle (template), cliquez sur Suivant « Next ». Dans notre cas, nous ne créons pas de projet à partir d'un modèle.



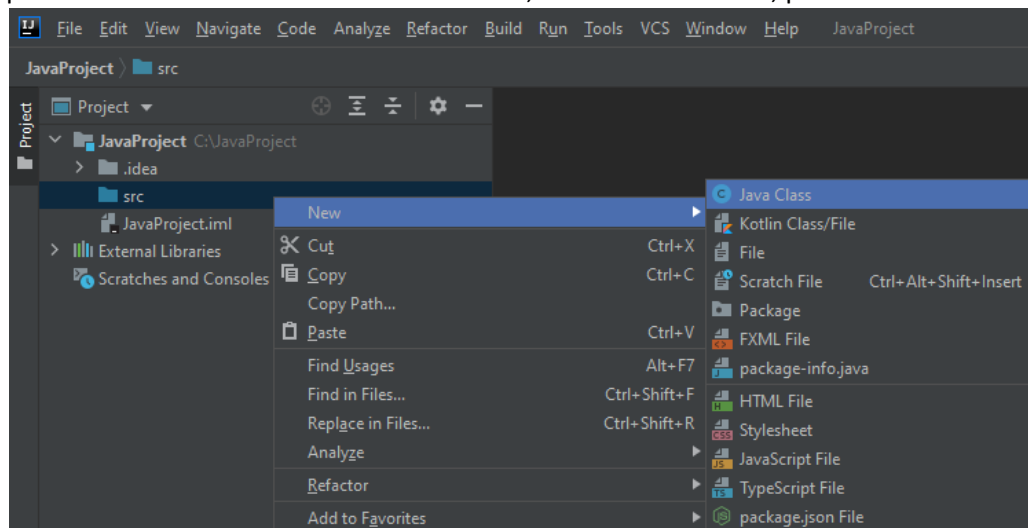
- 4) Enfin, choisissez un nom et un emplacement pour votre projet et cliquez sur Terminer « Finish ».



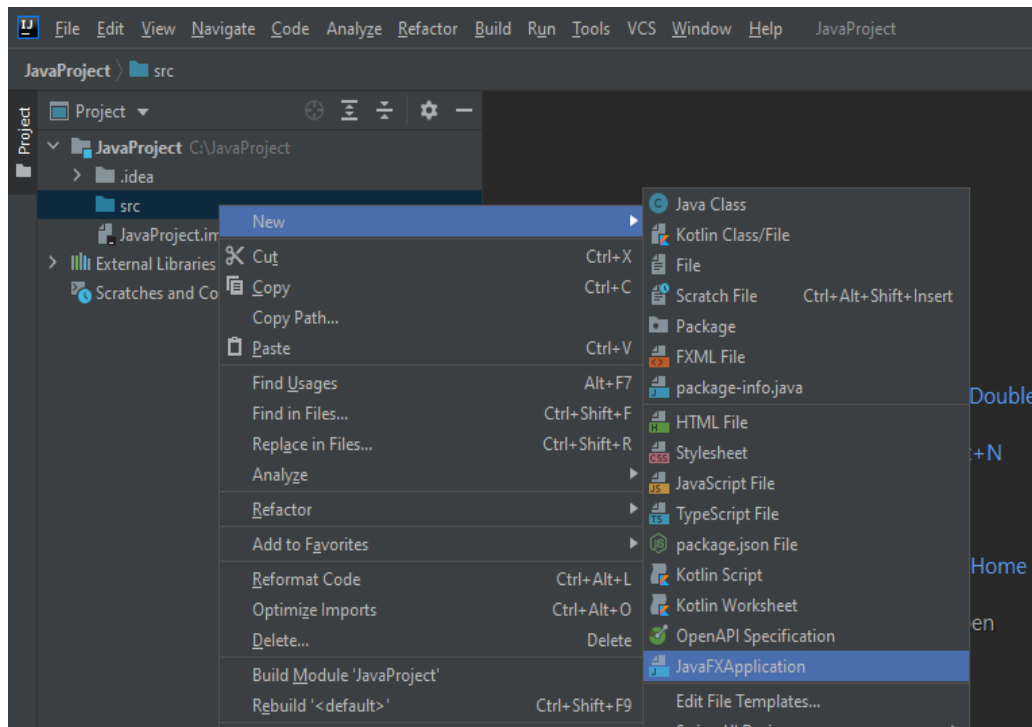


5) Pour créer une nouvelle classe :

Cliquez avec le bouton droit de la souris sur src, choisissez Nouveau, puis Classe Java.

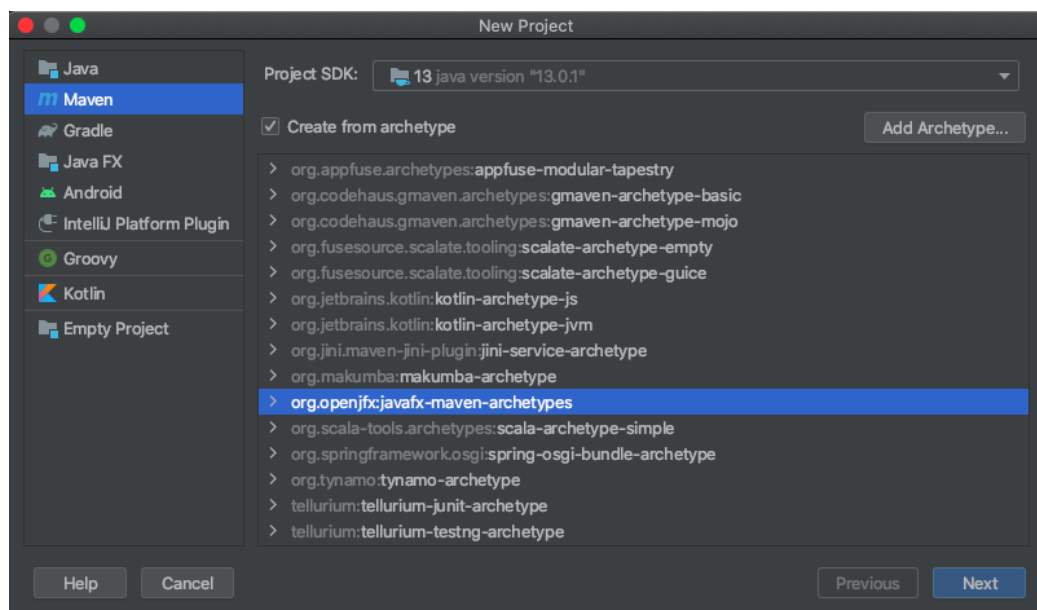


6) Pour créer une JavaFXApplication :

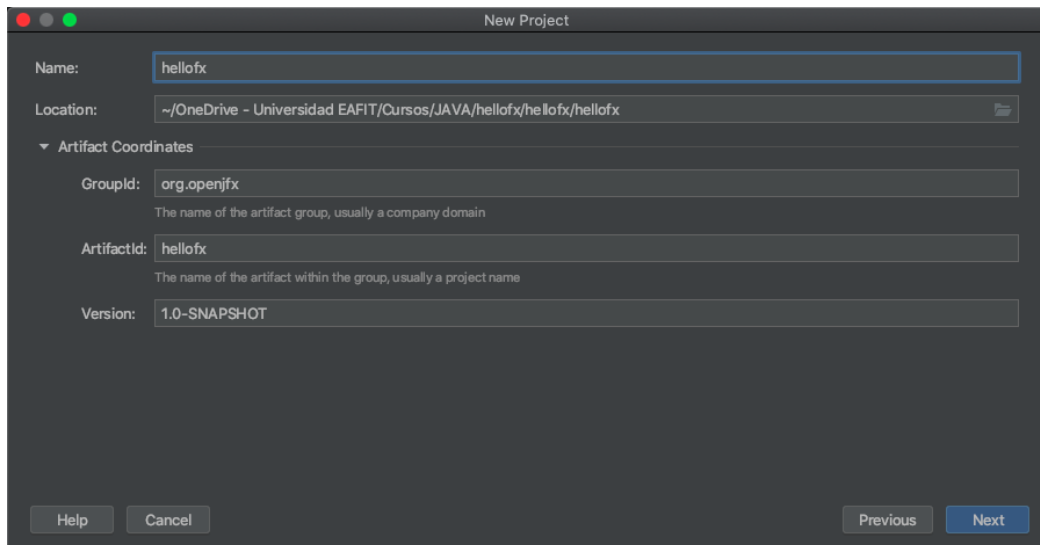


Si l'option JavaFXApplication n'est pas disponible :

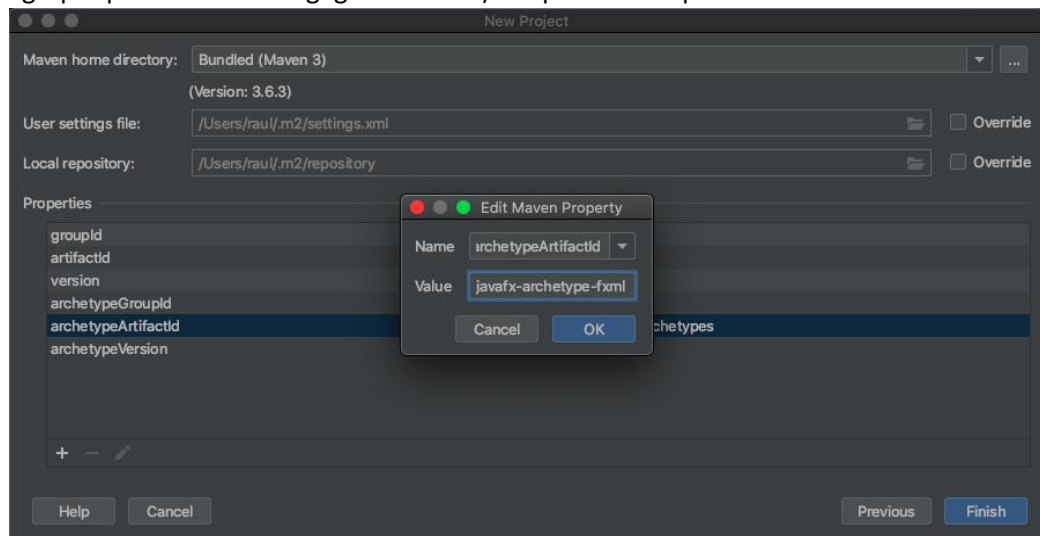
1. Téléchargez le dernier JavaFX SDK adapté à votre système d'exploitation : <https://gluonhq.com/products/javafx/>.
2. Décompressez l'archive et placez le dossier dans votre système de fichiers, par exemple : /Users/Desktop/javafx-sdk-15.0.1
3. Sélectionnez File -> New -> Project -> Maven et activez Create from archetype. Si l'archétype JavaFX n'est pas encore installé, sélectionnez Add archetype... et définissez le groupId (org.openjfx), l'artifactId (javafx-maven-archetypes) et la version (0.0.5), appuyez sur OK et sélectionnez l'artefact.



4. Fournissez le groupId, comme org.openjfx, l'artifactId, comme hellofx.



5. Dans Name, sélectionnez archetypeArtifactId. Dans le champ Value, saisissez javafx-archetype-simple (ou javafx-archetype-fxml si vous voulez faire des interfaces graphiques avec des langages à balises). Cliquez sur OK puis sur Finish.



6. Vérifiez que le pom.xml inclut les dépendances javafx.controls et javafx.fxml, et inclut le javafx-maven-plugin. Ouvrez la classe module-info, qui inclut les modules requis javafx.controls et javafx.fxml.

SI CETTE SOLUTION NE FONCTIONNE PAS :

1. Installez la dernière version de JAVA SE depuis le site web d'[Oracle](https://www.oracle.com/java/technologies/downloads/), vous devez exécuter le fichier exe : <https://www.oracle.com/java/technologies/downloads/>

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

Documentation Download

Linux **macOS** **Windows**

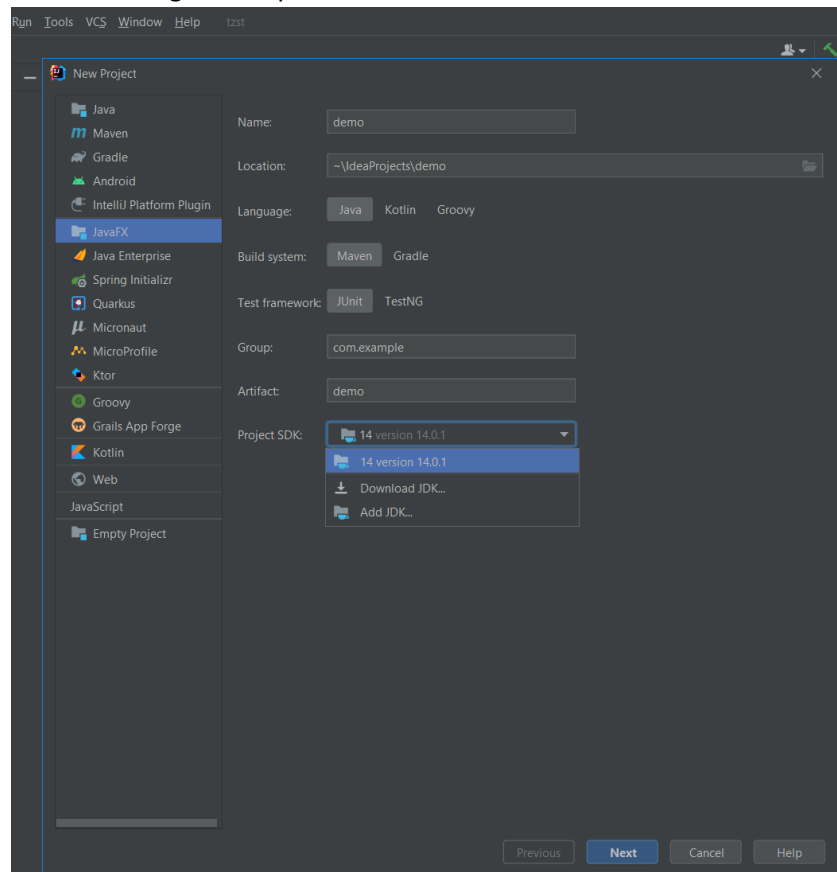
Product/file description	File size	Download
x64 Compressed Archive	170.64 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256 🔗)
x64 Installer	151.99 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256 🔗)
x64 MSI Installer	150.88 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256 🔗)

JDK 17 Script-friendly URLs

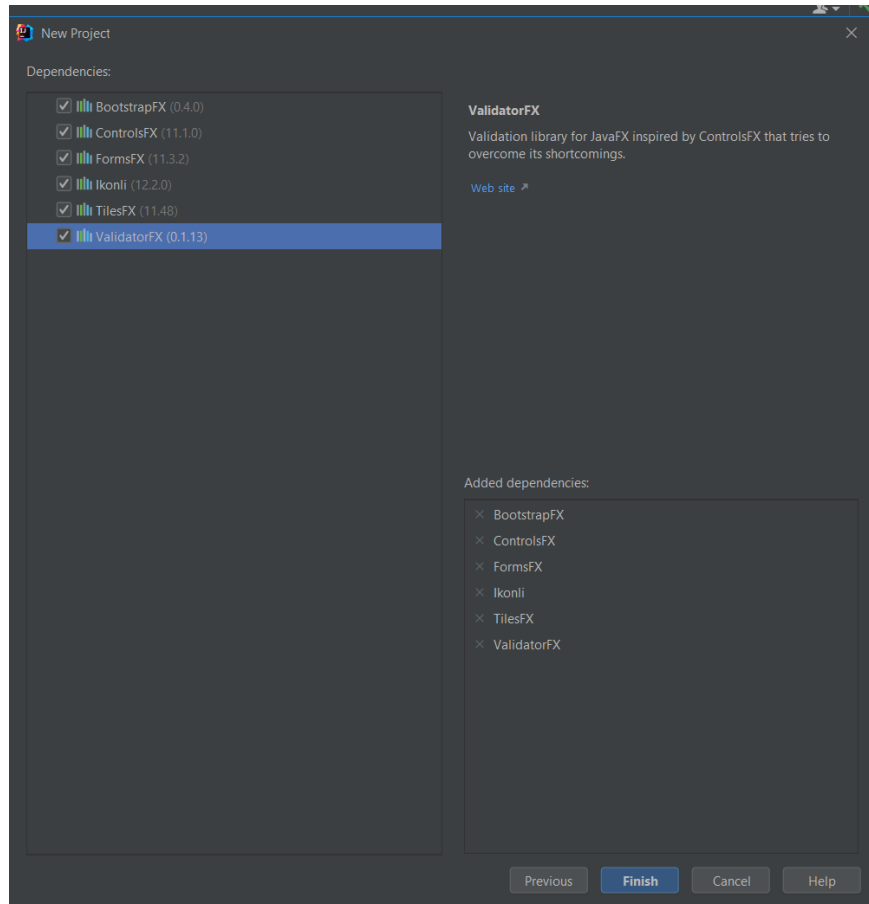
The URLs listed above will remain the same for all JDK 17 updates to allow their use in scripts.

[Learn more about automating the downloads of JDK 17](#)

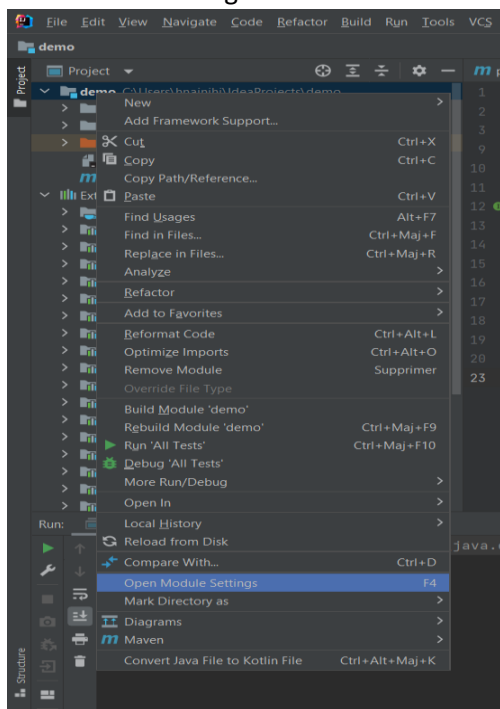
2. Redémarrez IntelliJ
3. Créez un nouveau projet JavaFx et définissez le SDK du projet comme celui que vous avez téléchargé à l'étape 1.



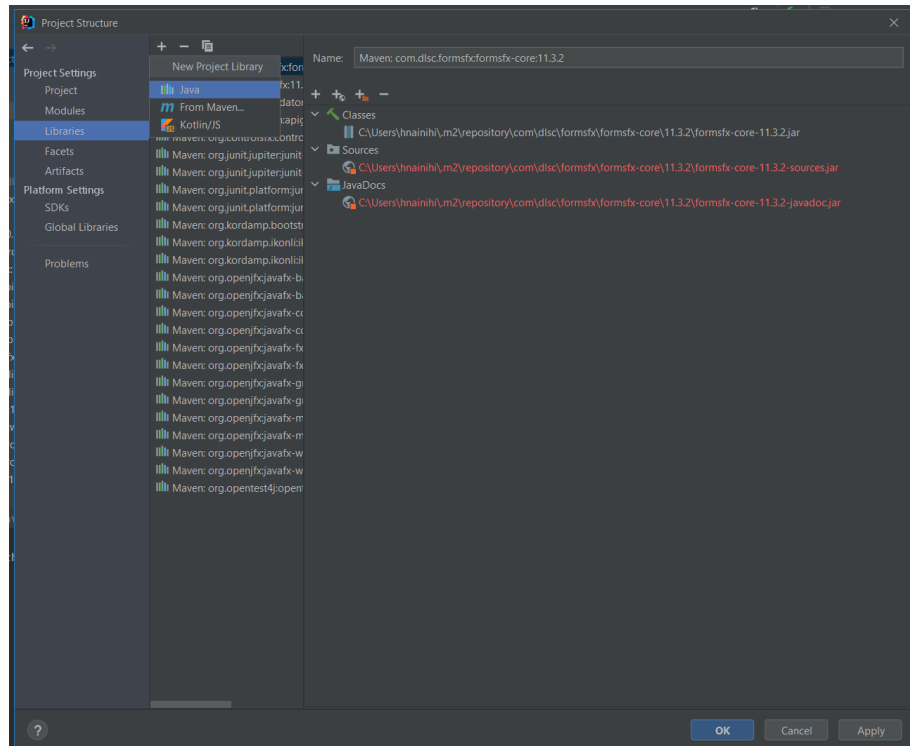
4. Sélectionnez toutes les dépendances puis sélectionnez finish



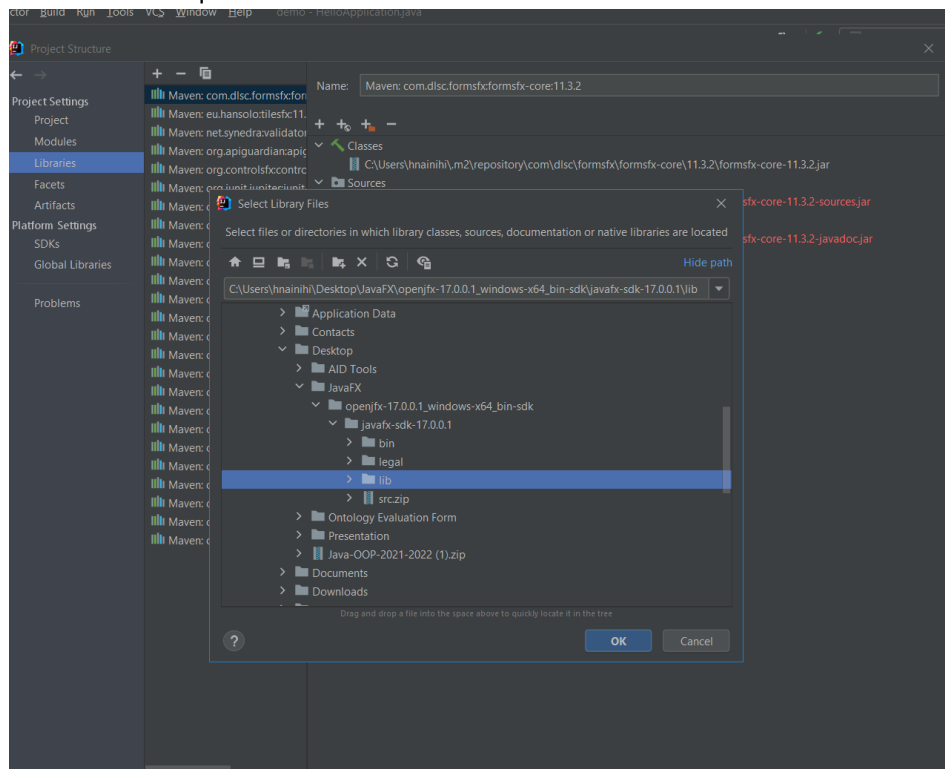
5. Essayez d'exécuter HelloApplication
6. Si cela ne fonctionne toujours pas, téléchargez le sdk javafx à partir de : <https://openjfx.io/> et extrayez-le.
7. Cliquez avec le bouton droit de la souris sur le nom du projet et sélectionnez Open Module Settings.



8. Allez dans Libraries et ajoutez une Java Project Library.

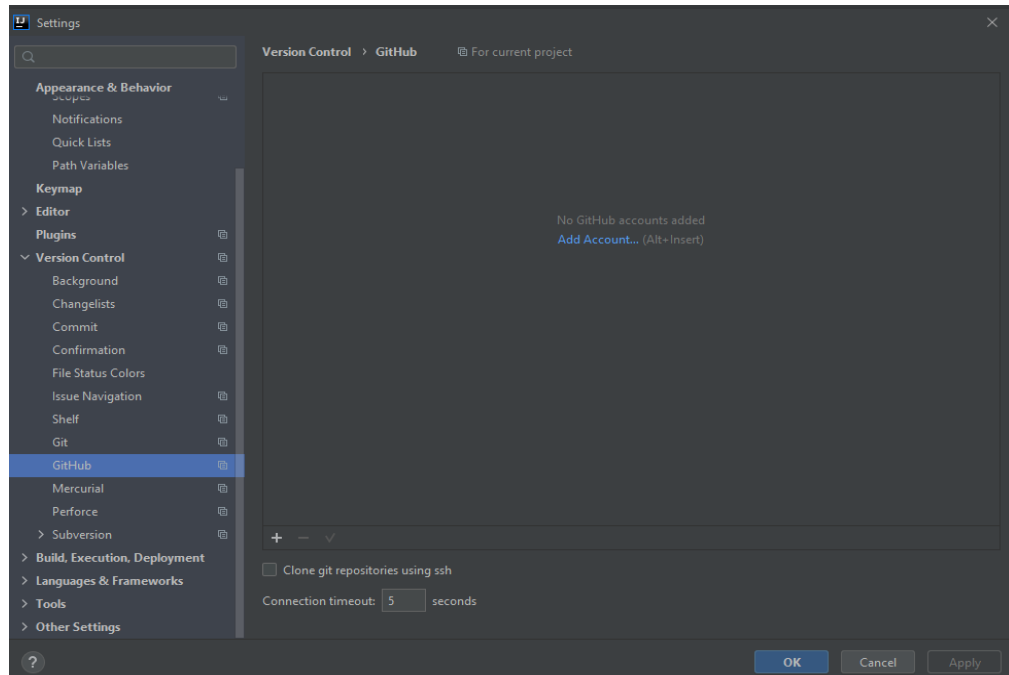


9. Trouvez le dossier que vous avez extrait (openjfx ...) puis choisissez le dossier lib à l'intérieur et cliquez sur OK.

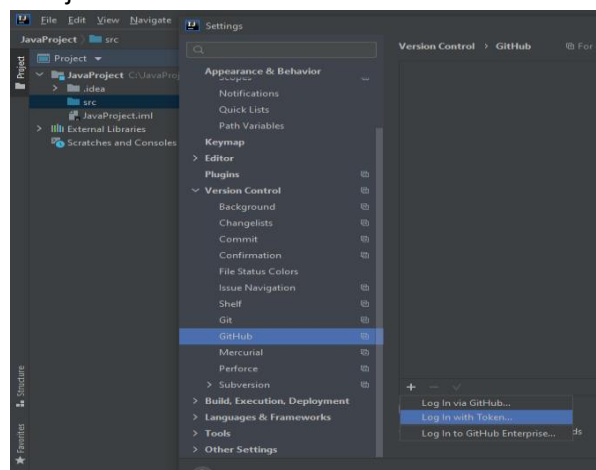


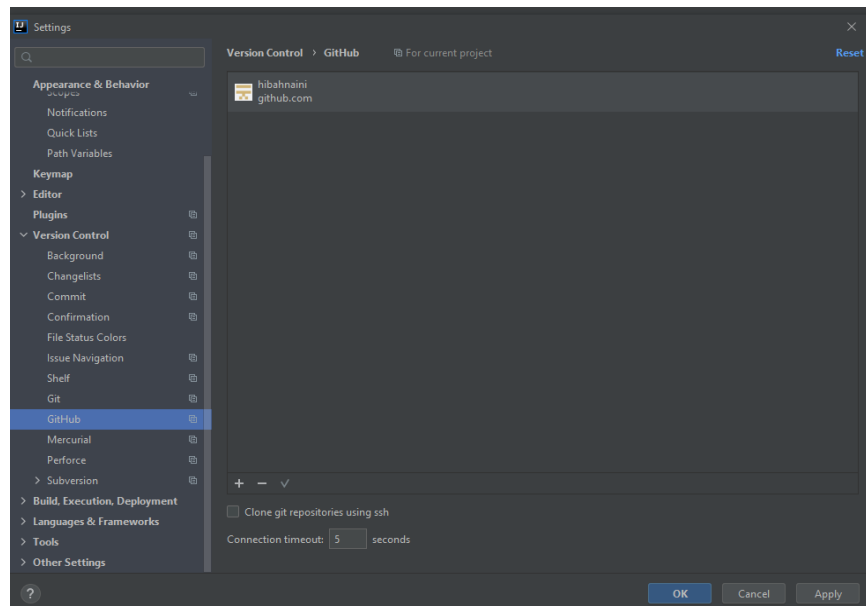
GitHub

- 1) Appuyez sur Ctrl+Alt+S pour ouvrir les paramètres de l'IDE et sélectionnez Contrôle de version (Version Control) | GitHub.

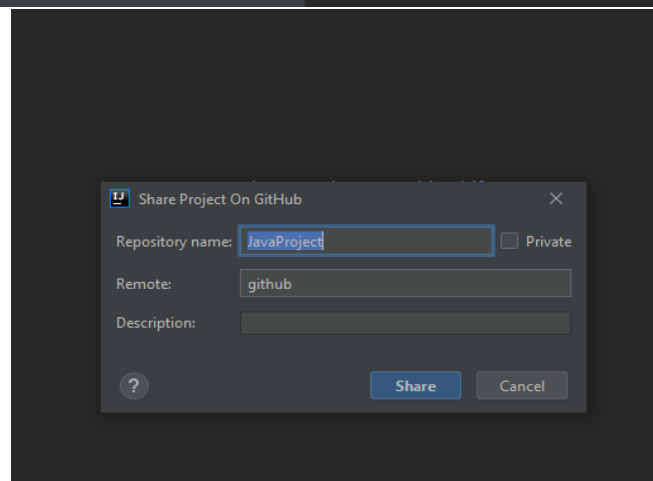
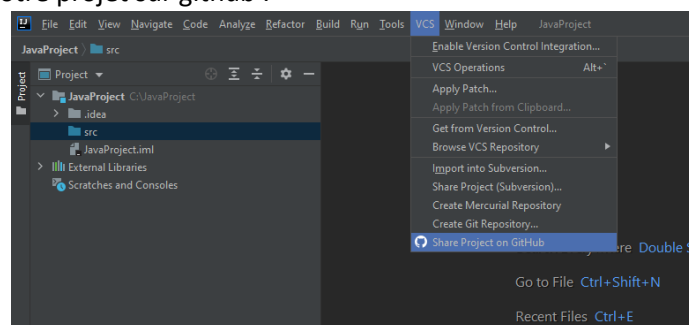


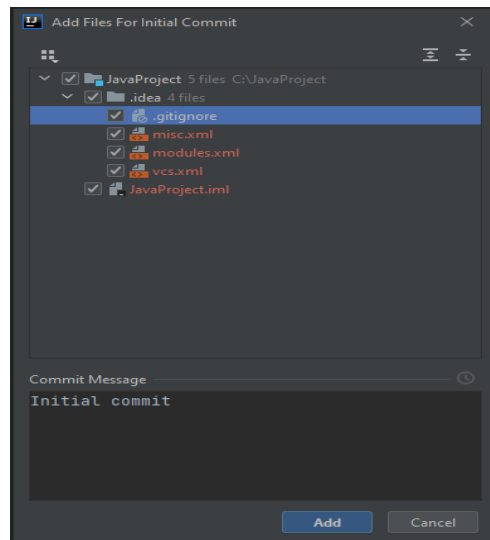
- 2) Cliquez sur le bouton Ajouter. (+)
- 3) Effectuez l'une des opérations suivantes :
 - Si vous avez déjà un jeton, cliquez sur le lien Utiliser le jeton (token) et collez-le à cet endroit.
 - Si vous souhaitez obtenir un nouveau token, saisissez votre identifiant et votre mot de passe. Si l'authentification à deux facteurs est activée, il vous sera demandé de saisir un code qui vous sera envoyé par SMS ou via l'application mobile. Consultez la rubrique Création d'un jeton d'accès personnel pour plus de détails sur les jetons GitHub.



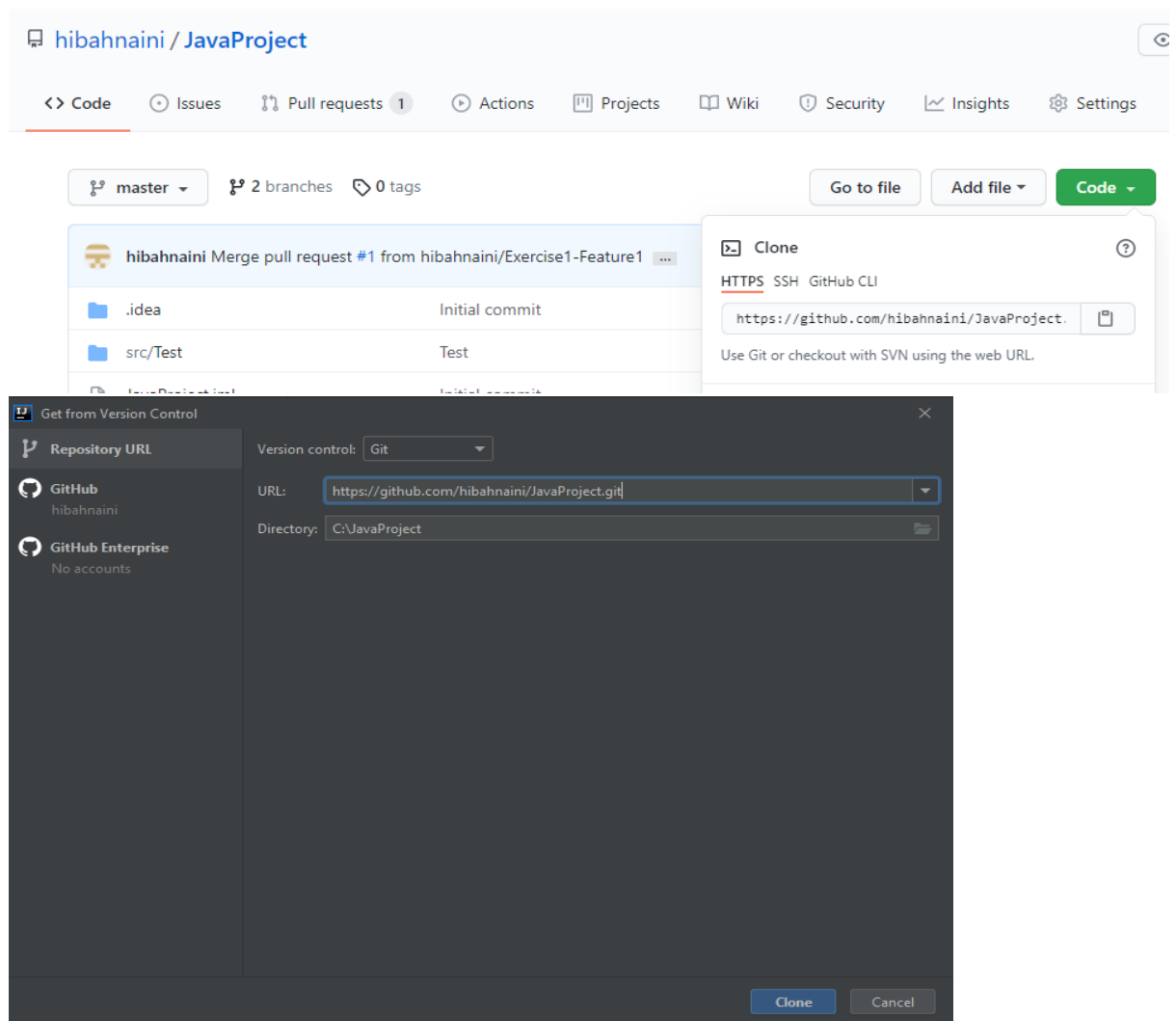


4) Pour partager votre projet sur github :

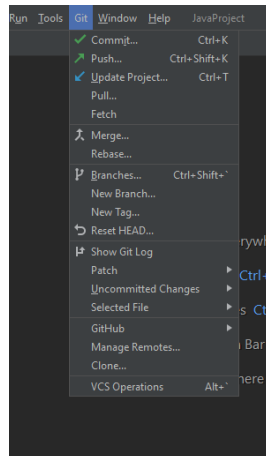




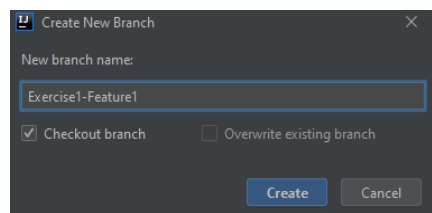
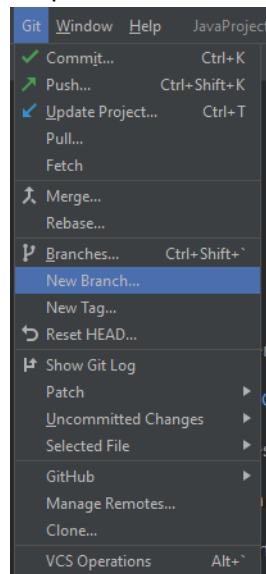
- 5) Pour les autres membres de l'équipe :
- Suivez les étapes 1, 2 et 3
 - VCS -> Get from Version Control
 - Copiez l'url du projet depuis github puis collez-la dans le champ URL, et cliquez sur clone.



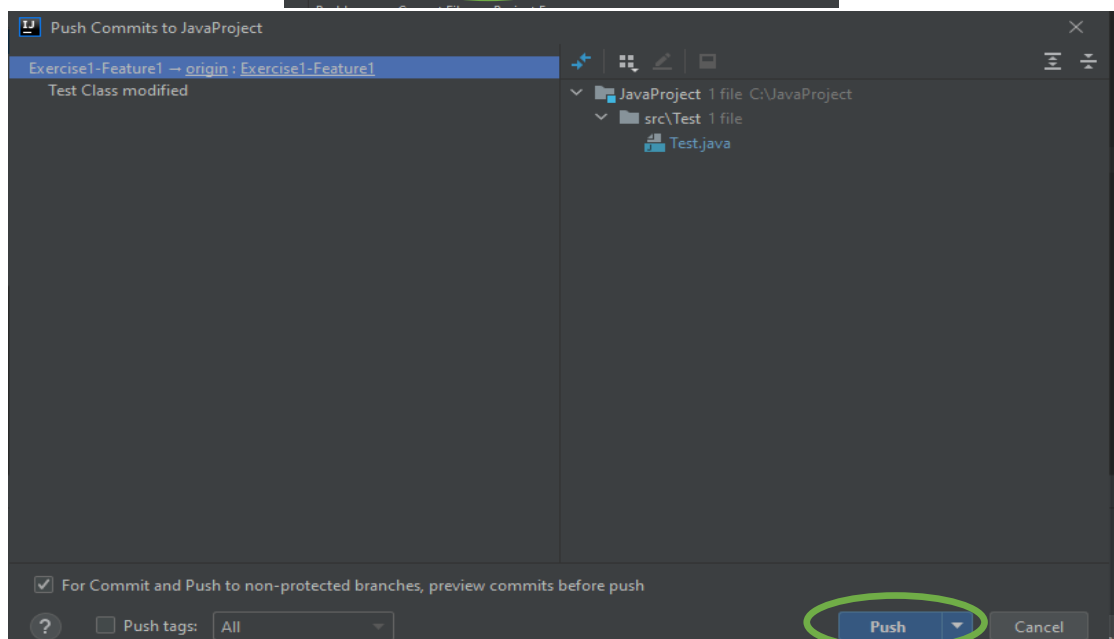
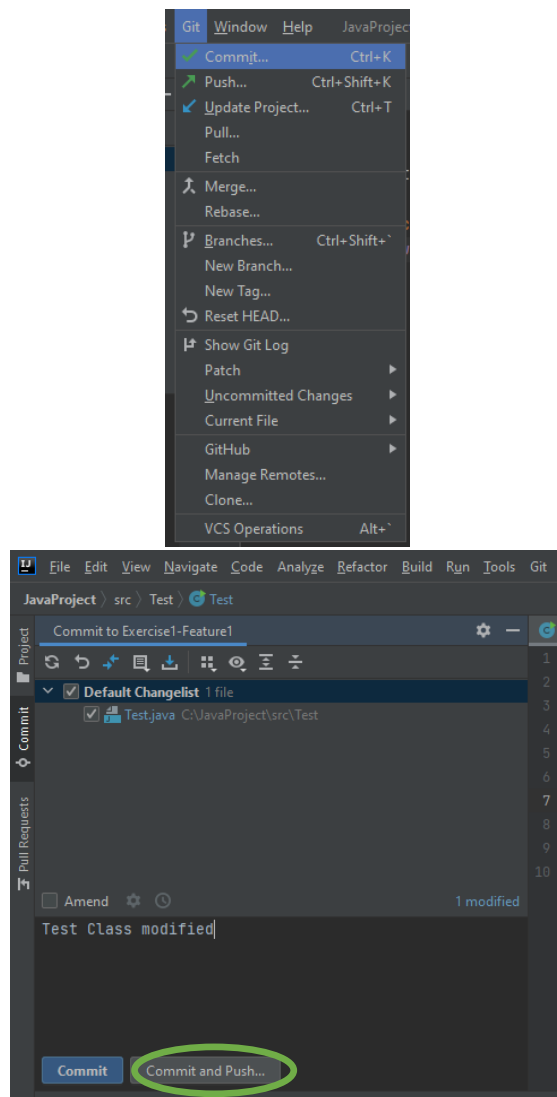
6) Vous devriez maintenant voir l'option Git dans le menu supérieur.



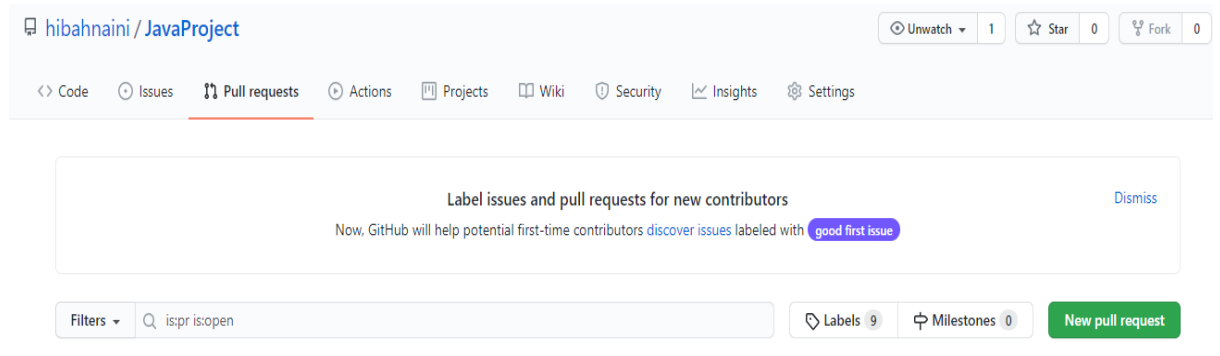
7) Créer une nouvelle branche pour chaque fonctionnalité (i.e. chaque partie de l'exercice)



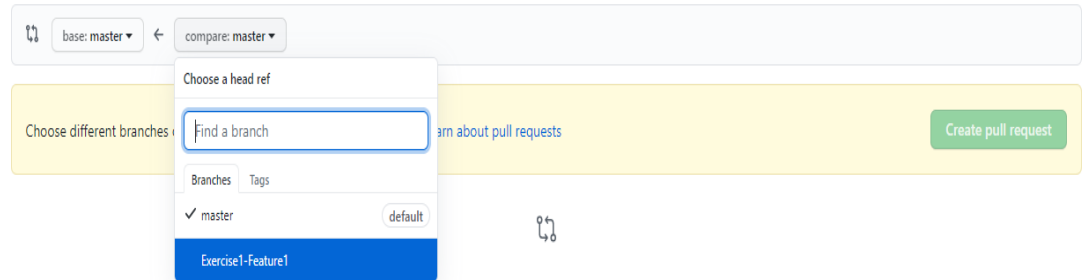
8) Commit et Poussez (Push) le code sur github une fois que vous avez fini de travailler.



- 9) Dans GitHub :
- Allez dans le dépôt du projet et cliquez sur "pull requests".



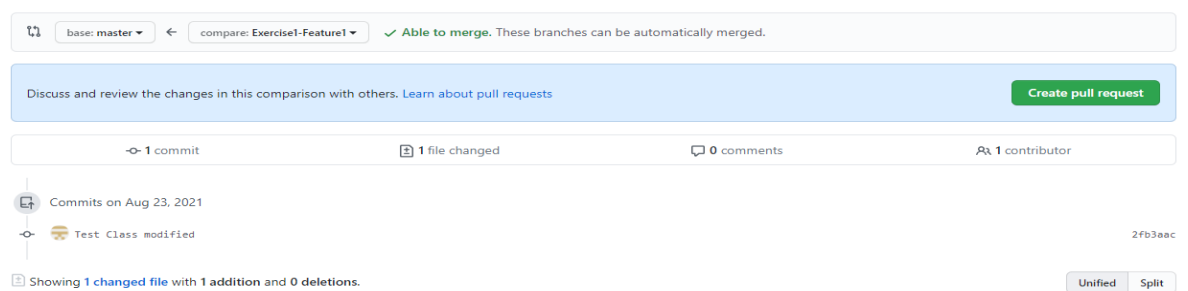
Ensuite, créez un "New Pull Request".



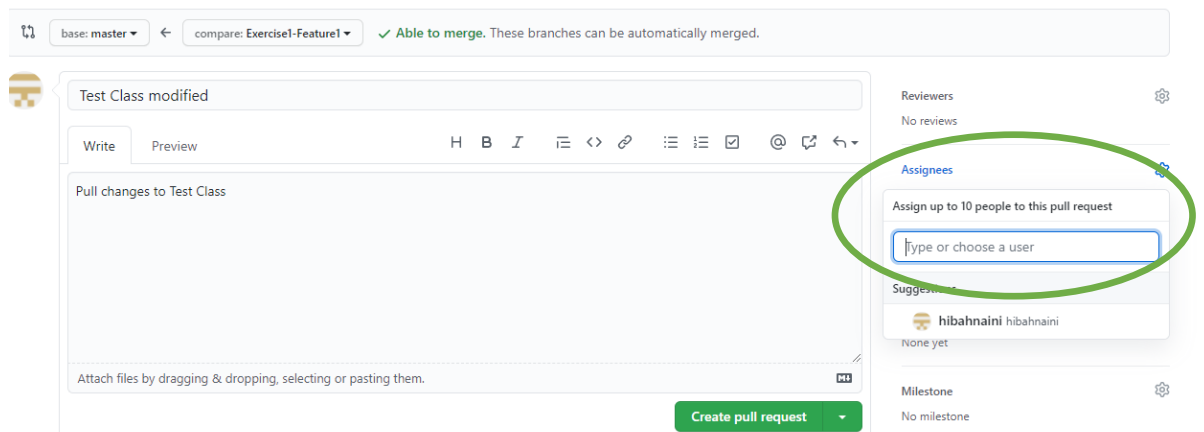
Choisissez votre branche comme branche à comparer à la base (master)

Comparing changes

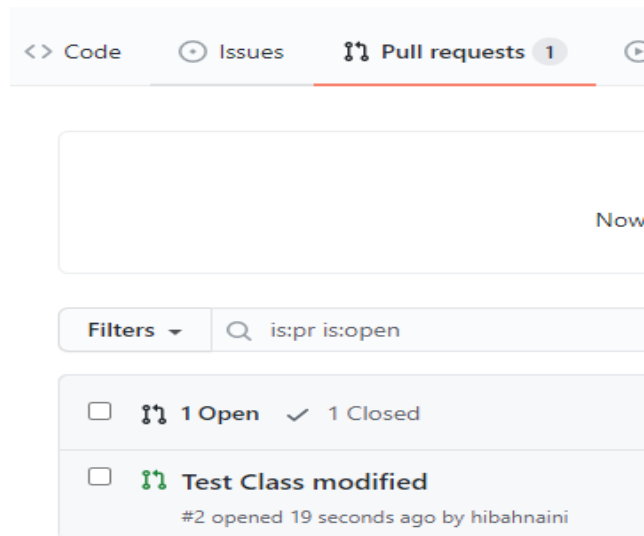
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



Créez un pull request pour votre commit





Ajoutez un commentaire à la demande de retrait et assignez-la à une autre personne ou à vous-même pour la fusionner (merge) avec la branche master.




Choisissez ensuite la demande de retrait dans Pull requests et fusionnez-la (merge).

Add more commits by pushing to the `Exercise1-Feature1` branch on `hibahnaini/JavaProject`.



**Continuous integration has not been set up**
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

**This branch has no conflicts with the base branch**
Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

1 Notions de base du langage et fonctions anonymes

1.1 No Pain, No Gain

Objectif pédagogique recherché avec cet exercice : apprendre à analyser et à corriger/améliorer un programme Java

Ce programme imprime un mot, en utilisant un générateur de nombres aléatoires pour sélectionner le premier caractère.

```
import java.util.Random;
public class Rhymes {
    private static Random rnd = new Random();
    public static void main(String[] args) {
        StringBuffer word = null;
        switch(rnd.nextInt(2)) {
            case 1: word = new StringBuffer('P');
            case 2: word = new StringBuffer('G');
            default: word = new StringBuffer('M');
        }
        word.append('a');
        word.append('i');
        word.append('n');
        System.out.println(word);
    }
}
```

Travail demandé :

- 1) Décrivez le comportement du programme.
- 2) Quel type de problèmes, d'erreurs ou de bugs peut avoir ce code et pourquoi ?
- 3) Comment éviter ces problèmes et comment améliorer (optimiser) ce code ? Écrivez une nouvelle version du code afin qu'il ne présente plus ces problèmes.
- 4) Êtes-vous sûr que la nouvelle version ne peut pas être améliorée davantage (plus générale, par exemple) ? Si oui, comment (vous pouvez utiliser la méthode de conception incrémentale vue en cours) ?

1.2 Un peu de récursivité

Objectif pédagogique recherché avec cet exercice : apprendre à utiliser le concept de récursivité en langage Java.

La récursivité est un concept qui fait référence à lui-même dans son fonctionnement. Cela se retrouve dans tous les champs artistiques : littérature (mise en abyme), peinture, photographie... Nous utilisons d'ailleurs tous les jours la récursivité lorsque nous définissons des mots ! En effet, nous utilisons des mots pour en définir d'autres, eux-mêmes étant définis par d'autres mots !

En programmation, il s'agit d'une fonction qui fait référence à elle-même. Deux fonctions peuvent s'appeler l'une l'autre, on parle alors de récursivité croisée.

Une fois n'est pas coutume, utilisons un exemple du monde des mathématiques : les factorielles. Une factorielle d'un entier naturel n est le produit des nombres entiers strictement positifs inférieurs ou égaux à n . Elle est notée $n!$ et se calcule ainsi : $n! = (n-1)! * n$.

Exemple : la factorielle de 10 est donc $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10 = 3\,628\,800$

Travail demandé

- 1) Écrire un programme contenant une méthode récursive permettant de calculer la factorielle d'un entier passé en paramètre. Utiliser le debugger de votre IDE pour exécuter la méthode pas à pas. Observer l'évolution de la pile d'appel.
- 2) Écrire un programme permettant de calculer le nème terme de la suite de Fibonacci. Utiliser une application « de force brute » de la formule de récurrence, et examiner l'évolution de la pile d'appel grâce au debugger. Essayer de trouver une solution récursive plus efficace (de complexité non exponentielle).
- 3) Écrire la fonction d'Ackermann définie par la formule suivante :
 - $\text{ack}(0,n) = n+1$
 - $\text{ack}(m,0) = \text{ack}(m-1,1)$
 - $\text{ack}(m,n) = \text{ack}(m-1, \text{ack}(m,n-1))$

Utiliser le debugger de votre IDE pour examiner l'évolution de la pile d'appel lors du calcul de $\text{ack}(2,2)$.

- 4) Écrire une méthode récursive permettant de calculer le plus grand commun diviseur (PGCD ; GCD en anglais) de deux entiers naturels par l'algorithme d'Euclide : étant donnés deux entiers naturels a et b , on commence par tester si b est nul. Si oui, alors le PGCD est égal à a . Sinon, on calcule c , le reste de la division de a par b . On remplace a par b , et b par c , et recommence le procédé.

1.3 Méthodes de tri

Objectif pédagogique recherché avec cet exercice : apprendre à écrire un programme java simple

1.3.1 Tri bulle (Bubble Sort)

Cette méthode de tri est fondée sur l'échange de deux éléments adjacents s'ils sont mal ordonnés. Un premier parcours du tableau permet de placer l'élément le plus grand à la fin et de réaliser un pré-tri dans la partie non triée. On recommence alors l'opération sur la partie non triée du tableau (*i.e.* les $n-1$ premiers éléments).

Exemple :

4	5	3	10	9	8	1	2
4	5	3	10	9	8	1	2
4	3	5	10	9	8	1	2
4	3	5	10	9	8	1	2
4	3	5	9	10	8	1	2
4	3	5	9	8	10	1	2
4	3	5	9	8	1	10	2
4	3	5	9	8	1	2	10

...

L'algorithme reprend donc pour classer les (n-1) éléments qui précèdent. L'algorithme se termine quand il n'y a plus de permutations possibles.

Travail demandé

1) Écrivez le programme BubbleSort dans un fichier BubbleSort.java

Pour initialiser le tableau contenant des valeurs, utiliser la méthode nextInt de la manière suivante :

```
// fill the array with random values
Random rand = new Random ();
for (int i = 0; i < tab.length; i++)
{
    // generate integers between 0 and 49 included
    tab [i] = rand.nextInt (50);
}
```

Pour pouvoir utiliser les générateurs de nombres aléatoires, ajouter en début de programme (avant le public class ...) la ligne suivante :

```
import java.util.*; // pour Random
```

Cet ajout sera expliqué dans la suite du cours.

1.3.2 Tri par insertion (Sort by Insertion)

Cet algorithme consiste en un découpage du tableau en une partie triée et une partie non triée, suivi d'une insertion de l'élément courant dans la partie du tableau triée.

Pour insérer l'élément courant, les éléments qui lui sont supérieurs sont décalés vers la droite.

Exemple :

4	5	3	10	9	8	1	2
4	5	3	10	9	8	1	2

3	4	5	10	9	8	1	2
3	4	5	10	9	8	1	2
3	4	5	9	10	8	1	2
3	4	5	8	9	10	1	2
1	3	4	5	8	9	10	2
1	2	3	4	5	8	9	10

Travail demandé :

- 1) Écrivez le programme InsSort dans un fichier InsSort.java
- 2) Écrivez un programme utilisant des méthodes de tri d'insertion et de tri bulle. Utilisez les fichiers précédents en refactorisant votre code. Nommez ce fichier Sort.java.

1.4 Température par ville

Objectif pédagogique recherché avec cet exercice : Apprendre à écrire et à lire une ligne spécifique dans un fichier.

Une station météorologique tient un registre des températures de chaque ville.

Le programme montre un menu de villes à l'employé chargé de la saisie des données. L'employé choisit la ville et saisit ensuite la température correspondante.

Le menu se présente comme suit :

Enter the number of the city :

0- Exit Program

1-Brest

2-Paris

3-Rennes

4-Toulouse

5-Nice

6-Lyon

Après avoir choisi la ville, le programme demande la température et l'enregistre avec la date et l'heure actuelles dans un fichier portant le nom de la ville.

Travail demandé :

- 1) Ecrivez le programme décrit ci-dessus dans le fichier Temp.java
 - Utilisez RandomAccessFile pour créer les fichiers
 - Écrivez le programme pour écrivez la température et la date dans chaque fichier.

2 Notions de base de la programmation orientée objet et les génériques de Java

2.1 Dépôt de voitures d'occasion (@The Australian National University)

Objectif pédagogique recherché avec cet exercice : Apprendre à spécifier et à utiliser une classe simple en Java.

Imaginez que vous dirigez un dépôt de voitures d'occasion et que vous voulez gérer un stock de voitures.

Crée une classe de voiture pour stocker des informations sur une voiture particulière :

Les champs de cette classe doivent être les suivants :

```
seriesNumber
manufacturer
manufactureDate
acquisitionDate
acquisitionCos
salePrice
condition
```

- 1) Faites-en sorte que la condition de champ soit une chaîne de trois valeurs possibles : "GOOD", "BAD", "AVERAGE".
- 2) Stockez les dates en utilisant String et définissez toutes les variables.
- 3) Écrivez aux constructeurs et obtenez et fixez des méthodes pour votre classe de voiture.
- 4) À partir d'une méthode principale dans une autre classe, remplissez un tableau d'inventaire de 6 voitures.
- 5) Imprimez votre inventaire. Rendez-le agréable à regarder (en utilisant la méthode `System.out.printf()`).
- 6) Appliquez une réduction de 10 % à toutes vos "mauvaises" voitures et imprimez à nouveau l'inventaire. (Notez que vous devrez peut-être ajouter des méthodes supplémentaires à votre catégorie de voiture).

2.2 Compte bancaire

Objectif pédagogique recherché avec cet exercice : Apprendre à spécifier et à utiliser une classe simple en Java.

Il s'agit de définir une classe permettant de modéliser des comptes bancaires. Cette classe (Compte) doit permettre à une application de créer et utiliser autant de comptes bancaires que nécessaires, chaque compte étant un objet, instance (ou exemplaire) de la classe Compte.

Un compte bancaire est identifié par un numéro de compte. Ce numéro de compte est un entier positif permettant de désigner et distinguer sans ambiguïté possible chaque compte géré par l'établissement

bancaire. Un compte est associé à une personne (civile ou morale) titulaire du compte, cette personne étant décrite par son nom. Une fois le compte créé, le titulaire du compte ne peut plus être modifié. La somme d'argent disponible sur un compte est exprimée en Euros. Cette somme est désignée sous le terme de solde du compte. Ce solde est un nombre décimal qui peut être positif, nul ou négatif.

Le solde d'un compte peut être éventuellement (et temporairement) être négatif. Dans ce cas, on dit que le compte est à découvert. Le découvert d'un compte est nul si le solde du compte est positif ou nul, il est égal à la valeur absolue du solde si ce dernier est négatif.

En aucun cas le solde d'un compte ne peut être inférieur à une valeur fixée pour ce compte. Cette valeur est définie comme étant - (moins) le découvert maximal autorisé pour ce compte. Par exemple pour un compte dont le découvert maximal autorisé est 2000 €, le solde ne pourra pas être inférieur à -2000 €. Le découvert maximal autorisé peut varier d'un compte à un autre, il est fixé arbitrairement par la banque à la création du compte et peut être ensuite révisé selon les modifications des revenus du titulaire du compte.

Créditer un compte consiste à ajouter un montant positif au solde du compte.

Débitier un compte consiste à retirer un montant positif au solde du compte. Le solde résultant ne doit en aucun cas être inférieur au découvert maximal autorisé pour ce compte. Lors d'une opération de débit, un compte ne peut être débité d'un montant supérieur à une valeur désignée sous le terme de débit maximal autorisé. Comme le découvert maximal autorisé, le débit maximal autorisé peut varier d'un compte à un autre et est fixé arbitrairement par la banque à la création du compte. Il peut être ensuite révisé selon les modifications des revenus du titulaire du compte.

Effectuer un virement consiste à débiter un compte au profit d'un autre compte qui sera crédité du montant du débit.

Lors de la création d'un compte seul le nom du titulaire du compte est indispensable. En l'absence de dépôt initial le solde est fixé à 0. Les valeurs par défaut pour le découvert maximal autorisé et le débit maximal autorisé sont respectivement de 800 € et 1000 €. Il est éventuellement possible d'attribuer d'autres valeurs à ces caractéristiques du compte lors de sa création.

Toutes les informations concernant un compte peuvent être consultées : numéro du compte, nom du titulaire, montant du découvert maximal autorisé, montant du débit maximal autorisé, situation du compte (est-il à découvert ?), montant du débit autorisé (fonction du solde courant, du débit maximal autorisé et du découvert maximal).

Travail demandé :

- 1) A partir du "cahier des charges" précédent, élaborer une spécification d'une classe Java modélisant un compte bancaire. Il s'agira en analysant le texte ci-dessus de :
 - définir les attributs (variables d'instance, variables de classe) de la classe Compte,
 - d'identifier les méthodes publiques proposées par la classe Compte. Pour chaque méthode on prendra soin, outre la définition de sa signature, de spécifier son comportement sous la forme d'un commentaire documentant.

- de proposer un ou plusieurs constructeurs pour la classe `Compte`. Là aussi on complétera la donnée de la signature de chaque constructeur avec un commentaire documentant détaillant son utilisation.

2) Réaliser une implémentation en langage Java de la classe précédemment spécifiée.

3) Rédigez la javadoc.

2.3 Taxes

Objectif pédagogique recherché avec cet exercice : Apprendre à transformer un programme donné en un programme orienté objet.

Le fichier `Taxes1.java` (dans le package `Taxes1` du projet `Taxes`) contient un programme qui aide une entreprise à stocker des données sur sa flotte de voitures et de calculer la taxe annuelle totale à payer pour la flotte. Le programme est modularisé sous forme de méthodes auxiliaires comme suite.

```
package Taxes1;

public class Taxes1 {
    public static void main(String[] args) {
        // Data of the first car (brand and cylinder capacity):
        String brand1 = "Toyota";
        int cylinderCapacity1 = 1598;
        // Data of the second car (brand and cylinder capacity):
        String brand2 = "BMW";
        int cylinderCapacity2 = 2756;
        System.out.println("FLEET AND TAXES OF THE ORIGINAL PROGRAM");
        // Display Fleet:
        displayCar(brand1, cylinderCapacity1);
        displayCar(brand2, cylinderCapacity2);
        // Calculation and display of total taxes:
        displayFleetTaxes(cylinderCapacity1, cylinderCapacity2);
    }

    static void displayCar(String brand, int cylinderCapacity) {
        // Display the information of the car
        System.out.println("You have a " + brand +
            " with a cylinder capacity of " + cylinderCapacity);
    }

    static void displayFleetTaxes(double cylinderCapacity1, double
cylinderCapacity2) {
        // calculate and display the total taxes of the two cars
        double taxes = calculateFleetTaxes(cylinderCapacity1,
cylinderCapacity2);
        System.out.println("Total taxes to pay : " + taxes + " euros");
    }

    static double calculateFleetTaxes(double cylinderCapacity1, double
cylinderCapacity2) {
        // Calculate the total taxes of the two cars
        double tax1 = calculateCarTaxes(cylinderCapacity1);
        double tax2 = calculateCarTaxes(cylinderCapacity2);
```

```

        return (tax1 + tax2);
    }
    static double calculateCarTaxes(double cylinderCapacity) {
        // return the tax to pay for a car with an indicated cylinder
capacity
        double tax;
        if (cylinderCapacity <= 1600)
            tax = 300.0;
        else if (cylinderCapacity <= 2300)
            tax = 500.0;
        else
            tax = 700.0;
        return tax;
    }
}

```

Travail demandé :

Transformer ce programme en un programme orienté objet. Pour réussir cette réingénierie, suivez les étapes suivantes :

1. Étudiez le fonctionnement du programme.
2. Créez une classe **Car** qui regroupera tout ce qui concerne un objet de type voiture, comme par exemple les variables contenant la marque et la cylindrée, la méthode pour afficher les données d'une voiture et la méthode pour calculer la taxe d'une voiture. Les méthodes d'instance dans le nouveau programme auront typiquement moins de paramètres que les méthodes auxiliaires correspondantes dans l'ancien programme. N'oubliez pas qu'une méthode d'instance n'a pas de mot-clé **static**. Ajoutez une méthode constructeur à la classe et le mot-clé **private** pour les variables et les méthodes d'instance qui ne seront pas utilisées à l'extérieur de la classe.
3. Créez une classe **Fleet** qui regroupera tout ce qui concerne un objet de type flotte, e.g. les variables pour les voitures, la méthode pour afficher les données d'une flotte et la méthode pour calculer la taxe d'une flotte. Ces méthodes appelleront typiquement les méthodes de la classe **Car**.
4. Modifiez la méthode **main**. Idéalement, elle n'aura que 3 instructions : construction d'un objet de type **Fleet**, appel à une méthode d'instance pour afficher les informations sur la flotte et appel à une méthode d'instance pour calculer et afficher la taxe totale de la flotte.

Exemple d'exécution du programme demandé :

You have a Toyota with a cylinder capacity of 1598

You have a BMW with a cylinder capacity of 2756

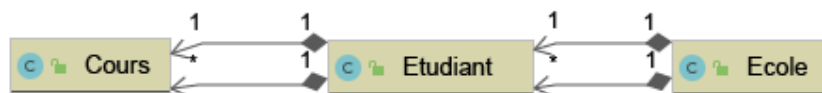
Total tax to pay : 1000.0 euros

2.4 Dossiers des étudiants

Objectif pédagogique recherché avec cet exercice : apprendre à spécifier et à utiliser une classe simple en Java (Liste des objets)

Une école tient un registre de toutes les informations concernant ses élèves. Ces informations comprennent le prénom, le nom de famille, la date de naissance, le numéro de téléphone et l'adresse électronique des parents, l'année d'études et la section de l'élève.

Chaque étudiant dispose d'une liste de cours. Chaque cours a un identifiant, un nom, un professeur, une note, un nombre d'heures.



Travail demandé :

1. Créer les 3 classes : Ecole, Cours et Etudiant
2. Définissez les variables de chaque classe comme privées.
3. Définir les getters et setters de chaque variable
4. Créer deux constructeurs de cours, un avec seulement l'ID du cours et un autre avec toutes les variables
5. Créer deux constructeurs de Etudiant, un avec toutes les variables sauf la liste des cours, et un autre qui inclut la liste des cours
6. Créer la méthode ajouterCours dans la classe Etudiant qui ajoute un cours à la liste des cours d'un étudiant
7. Créer la méthode supprimerCours qui supprime un cours de la liste des cours d'un étudiant
8. Créer la méthode ajouterEtudiant qui ajoute un élève à la liste des élèves de l'école
9. De même, créez ajouterEtudiant et supprimerEtudiant dans la classe Ecole
10. Créer la méthode moyenne qui génère la moyenne de l'étudiant dans tous les cours
11. Dans la classe Ecole, créez une méthode de classe tousEtudiants qui montre tous les élèves de l'école
12. Dans le programme principal, montrer la liste de tous les étudiants, puis la moyenne de 2 étudiants (avec leurs noms)

2.5 Cercles

Objectif pédagogique recherché avec cet exercice : apprendre à compléter un programme donné en un programme orienté objet

```
package shapes;

public class Circle {

    private double radius;
    private double x;
```

```

private double y;

public double area() {
    return radius * radius * Math.PI;
}

public double perimeter() {
    return 2 * Math.PI * radius;
}

public void position() {
    System.out.printf("Position of the cricle is (%.1f,%.1f)\n", x, y);
}

public double getRadius() {
    return radius;
}

public void setRadius(double radius) {
    if (radius > 0) {
        this.radius = radius;
    }
}

public double getX() {
    return x;
}

public void setX(double x) {
    this.x = x;
}

public double getY() {
    return y;
}

public void setY(double y) {
    this.y = y;
}

public Circle(double radius) {
    this.radius = radius;
}

public Circle(double radius, double x, double y) {
    this.radius = radius;
    this.x = x;
    this.y = y;
}
}

```

Travail demandé :

- 1) Ajoutez une méthode publique distanceToOrigin() à la classe Circle.
La méthode retourne la distance entre le point central du cercle et l'origine point (0,0, 0,0).
- 2) Ensuite, écrivez un programme Java pour effectuer les tâches suivantes :
 - Générer un nombre aléatoire N dans l'intervalle [5, 10].
 - Créer N cercles. Chaque cercle a un rayon aléatoire dans la plage [1.0, 3.0] et un position centrale aléatoire : x et y sont dans la plage [2.0, 5.0].
 - Parmi les cercles générés, trouvez celui qui a la plus petite surface et celui dont le centre est le plus éloigné du point d'origine.

Pour la génération de nombres aléatoires, vous pouvez utiliser les deux méthodes suivantes de la Classe aléatoire :

(1) public int nextInt(int bound)

(2) public double nextDouble()

Voir <https://docs.oracle.com/javase/8/docs/api/java/util/Random.html> pour plus de détails.

Un exemple de sortie :

Circle #1: radius = 2.14, x = 2.00, y = 4.04

Circle #2: radius = 1.45, x = 3.33, y = 4.16

Circle #3: radius = 1.89, x = 4.68, y = 4.87

Circle #4: radius = 1.59, x = 4.97, y = 4.47

Circle #5: radius = 2.71, x = 2.66, y = 4.83

Circle #6: radius = 2.42, x = 2.18, y = 3.34

Circle #7: radius = 2.72, x = 4.60, y = 2.46

Circle #2 is the smallest circle, area = 6.62

Circle #3 is the farthest circle, distance to origin = 6.75

3 Notions avancées de la POO

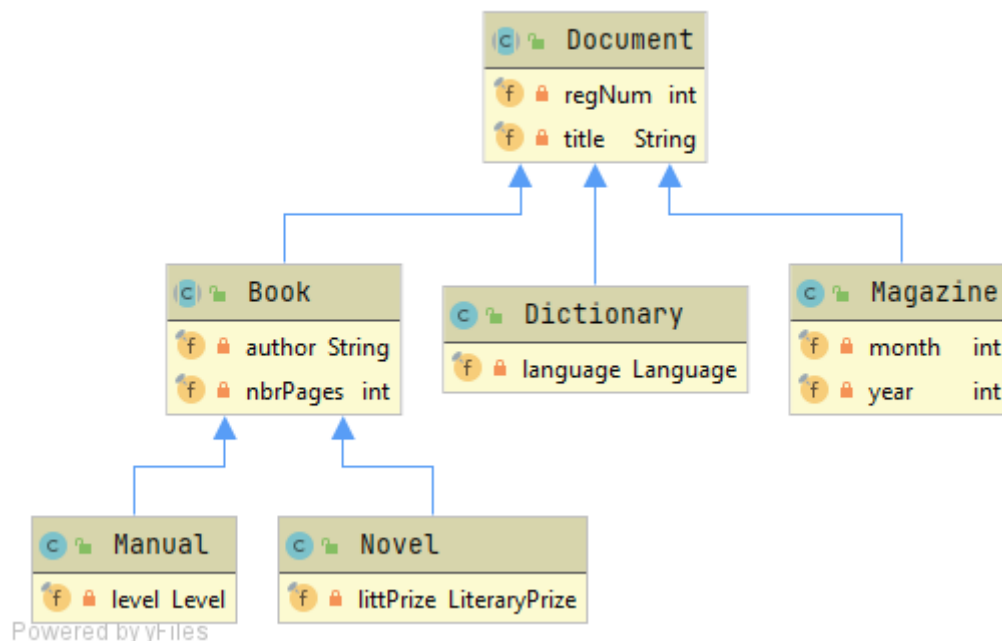
3.1 Bibliothèque (inspiré des TPs de Laurent Tichit)

Objectif pédagogique recherché avec cet exercice : apprendre à utiliser la notion d'héritage en Java

Pour la gestion d'une bibliothèque on nous demande d'écrire une application traitant des *documents* de nature diverse : des *livres*, qui peuvent être des *romans* ou des *manuels*, des *revues*, des *dictionnaires*, etc.

Tous les documents ont un numéro d'enregistrement (un entier) et un titre (une chaîne de caractères). Les livres ont, en plus, un auteur (une chaîne) et un nombre de pages (un entier). Les romans ont éventuellement un prix littéraire (une énumération ou un entier conventionnel, parmi : **GONCOURT**, **MEDICIS**, **INTERALLIE**, etc.), tandis que les manuels ont un niveau scolaire (un entier ou une énumération **ECOLE_ELEMENTAIRE**, **COLLEGE**, **LYCEE**). Les revues ont un mois et une année (des entiers) et les dictionnaires ont une langue (une énumération, etc.).

Tous les divers objets en question ici (*livres*, *revues*, *dictionnaires*, *romans*, etc.) doivent pouvoir être manipulés en tant que *documents*.



Travail demandé :

- 1) Définissez les classes **Document**, **Livre**, **Roman**, **Manuel**, **Revue** et **Dictionnaire**, entre lesquelles existeront les liens d'héritage que la description précédente suggère.

Dans chacune de ces classes définissez

- le constructeur qui prend autant d'arguments qu'il y a de variables d'instance et qui se limite à initialiser ces dernières avec les valeurs des arguments,
- une méthode **public String toString()** produisant une description sous forme de chaîne de caractères des objets,
- si vous avez déclaré **private** les variables d'instance (c'est conseillé, sauf indication contraire), définissez également des « accesseurs » publics *get...* permettant de consulter les valeurs de ces variables.

Écrivez une classe exécutable **TestDocuments** qui crée et affiche plusieurs documents de types différents.

Exemple de la méthode **main** du programme :

```
public static void main(String[] args) {
    Roman leRougeEtLeNoir = new Roman(1,
        "Le rouge et le noir",
        "Stendhal",
        563,
        null);
    Roman leursEnfantsApresEux = new Roman(2,
        "Leurs enfants après eux",
        "Nicolas Matthieu",
        354,
        Roman.PrixLitteraire.GONCOURT);
    Manuel histoire6eme = new Manuel(7,
        "Histoire-Géographie-EMC 6e",
        "Émilie Blanchard",
        154,
        Manuel.Niveau.COLLEGE);
    Revue geoJuin2019 = new Revue(56,
        "Géo",
        6,
        2019);
    Dictionnaire harrapsDePoche = new Dictionnaire(82,
        "Harrap's de poche",
        Dictionnaire.Langue.ANGLAIS);

    System.out.println("leRougeEtLeNoir="+leRougeEtLeNoir.toString());
    System.out.println("leursEnfantsApresEux="+leursEnfantsApresEux.toString());
    ;
    System.out.println("histoire6eme="+histoire6eme.toString());
    System.out.println("geoJuin2019="+geoJuin2019.toString());
    System.out.println("harrapsDePoche="+harrapsDePoche.toString());
}
```

- 2) Définissez une classe **Bibliotheque** qui sera représentée par un *tableau de documents* ou encore mieux, par une **ArrayList** et qui comportera les méthodes présentées ci-dessous. Si vous ne l'avez pas encore fait, jetez un œil sur la documentation de la classe **java.util.ArrayList**. Une **ArrayList AL** se comporte comme un tableau **T**, c'est-à-dire qu'elle offre l'accès indexé optimisé (on dit « en temps constant ») à ses éléments, sauf qu'au lieu de **x = T[i]** il faut écrire **x = AL.get(i)** et au lieu de **T[i]= x** il faut écrire **AL.set(i,x)** ou juste **AL.add(x)** pour qu'elle gère automatiquement l'indexation de l'élément **x** (voir slides "Beyond the basics" de la section "Notions de base du langage Java" du

cours). Cependant, une **ArrayList** a un avantage considérable sur un tableau : elle s'occupe de l'allocation de son espace mémoire, l'augmentant lorsque c'est nécessaire, sans que le programmeur ait à s'en soucier.

- **Bibliotheque(int capacite)** - constructeur qui crée une bibliothèque ayant la capacité (nombre maximum de documents) indiquée,
- **void afficherDocuments()** - affiche *tous* les ouvrages de la bibliothèque,
- **Document document(int i)** - renvoie le *i*^{ème} document,
- **boolean ajouter(Document doc)** - ajoute le document indiqué et renvoie **true** (**false** en cas d'échec),
- **boolean supprimer(Document doc)** - supprime le document indiqué et renvoie **true** (**false** en cas d'échec)
- **void afficherAuteurs()** - affiche la liste des auteurs de tous les ouvrages qui ont un auteur (au besoin, utilisez l'opérateur **instanceof**)

Exemple de la méthode **main** du programme avec une **Bibliotheque** :

```
public static void main(String[] args) {
    Bibliotheque maBibliotheque = new Bibliotheque(10);
    maBibliotheque.ajouter(new Roman(1,
        "Le rouge et le noir",
        "Stendhal",
        563,
        null));
    maBibliotheque.ajouter(new Roman(2,
        "Leurs enfants après eux",
        "Nicolas Matthieu",
        354,
        Roman.PrixLitteraire.GONCOURT));
    maBibliotheque.ajouter(new Manuel(7,
        "Histoire-Géographie-EMC 6e",
        "Émilie Blanchard",
        154,
        Manuel.Niveau.COLLEGE));
    maBibliotheque.ajouter(new Revue(56, "Géo", 6, 2019));
    maBibliotheque.ajouter(new Dictionnaire(82,
        "Harrap's de poche", Dictionnaire.Langue.ANGLAIS));
    maBibliotheque.afficherDocuments();
    maBibliotheque.afficherAuteurs();
}
```

Le code demandé doit produire un résultat comme celui-ci :

```
Roman{numEnreg=1,titre='Le rouge et le
noir',auteur='Stendhal',nbrPages=563,prixLitt=null}
```

```
Roman{numEnreg=2,titre='Leurs enfants après eux',auteur='Nicolas
Matthieu',nbrPages=354,prixLitt=GONCOURT}
```

```
Manuel{numEnreg=7,titre='Histoire-Géographie-EMC 6e',auteur='Émilie
Blanchard',nbrPages=154,niveau=COLLEGE}
```

```
Revue{numEnreg=56,titre='Géo',mois=6,annee=2019}
```

```
Dictionnaire{numEnreg=82,titre='Harrap's de poche',langue=ANGLAIS}
```

Stendhal

Nicolas Matthieu

Émilie Blanchard

Définissez, avec un effort minimal, une classe **Livrotheque** dont les instances ont les mêmes fonctionnalités que les **Bibliothèques** mais sont *entièrement constituées de livres*. Comment optimiser dans la classe **Livrotheque** la méthode **afficherAuteurs** ?

Exemple de la méthode **main** du programme avec le changement demandé :

```
public static void main(String[] args) {
    Livrotheque maBibliothèque = new Livrotheque (10);
    maBibliothèque.ajouter(new Roman(1,
        "Le rouge et le noir",
        "Stendhal",
        563,
        null));
    maBibliothèque.ajouter(new Roman(2,
        "Leurs enfants après eux",
        "Nicolas Matthieu",
        354,
        Roman.PrixLitteraire.GONCOURT));
    maBibliothèque.ajouter(new Manuel(7,
        "Histoire-Géographie-EMC 6e",
        "Émilie Blanchard",
        154,
        Manuel.Niveau.COLLEGE));

    maBibliothèque.afficherDocuments();
    maBibliothèque.afficherAuteurs();
}
```

Évidemment, le code doit produire le même résultat.

- 3) (**Pour aller plus loin**) Il est possible aussi de rendre la classe **Bibliothèque** générique, avec un type générique héritant de **Document**, et de sous-classer cette bibliothèque en tant que **Livrotheque** en précisant ce type générique. Modifier votre programme en ce sens.

3.2 Les polygones

Objectif pédagogique recherché avec cet exercice : mettre en pratique le concept d'héritage en Java

En géométrie euclidienne, un polygone (du grec *polus*, nombreux, et *gônia*, angle) est une figure géométrique plane formée d'une ligne polygonale fermée, c'est-à-dire d'une suite cyclique de segments consécutifs.

Un polygone est constitué :

- d'une suite finie² de points du plan appelés sommets³ ;
- des segments reliant les couples de sommets consécutifs ainsi que d'un segment reliant le premier et le dernier point, tous ces segments étant appelés côtés.

Travail demandé :

- 1) Écrire une classe **Point** pour stocker les coordonnées d'un point dans le plan :
 - écrire un constructeur qui accepte **x** et **y** (**double**)
 - écrire les accesseurs **getX** et **getY**
 - ajouter une méthode **distance**, calculer la distance par rapport à un autre point
 - écrire une méthode statique **meReel** qui accepte 2 **double** et retourne vrai s'il sont égaux à un ϵ près (par exemple $1e-10$). On l'utilisera toujours pour comparer 2 réels ¹
 - redéfinir la méthode **equals** pour tester 2 points (utiliser **meReel**)
 - redéfinir la méthode **toString** pour qu'elle renvoie « <x,y> »

- 2) Écrire la classe **Polygone** qui représente un polygone sous forme d'un tableau de **Point** (les sommets du polygone) :
 - écrire un constructeur acceptant un tableau de **Point**. Ajouter des constructeurs si nécessaire
 - écrire l'accesseur **getSommets**
 - écrire une méthode **perimetre** qui retourne le périmètre du polygone
 - écrire une méthode **surface** qui retourne **-1** (voulant dire qu'on ne sait pas calculer une telle surface). Vous pourrez l'implémenter plus tard
 - écrire une méthode **texteSommets** qui retourne une chaîne contenant la liste des sommets
 - redéfinir la méthode **equals** pour qu'elle teste vraiment si on a 2 mêmes polygones
 - redéfinir la méthode **toString** pour renvoyer « polygone à N sommets » et ses sommets.

- 3) Ecrire une classe **Triangle** qui hérite de **Polygone** :
 - Ecrire le constructeur qui prend 3 **Point**.
 - Redéfinir la méthode **surface**. On utilisera la formule de Héron d'Alexandrie qui fournit la surface d'un triangle en fonction des longueurs de chaque côté (a, b, c) :

$$\text{surface} = \sqrt{p \times (p-a) \times (p-b) \times (p-c)} \quad \text{où } p \text{ est le demi-périmètre du triangle.}$$

¹ Evidemment, pas tous les réels peuvent être codés (même en double). Du coup l'arithmétique flottante est piégeuse et sujette à arrondis et résultats imprévisibles. Essayez ce code : `double x = 1.2 * 3.0; if (x != 3.6) System.out.println("ERREUR: " + x);`

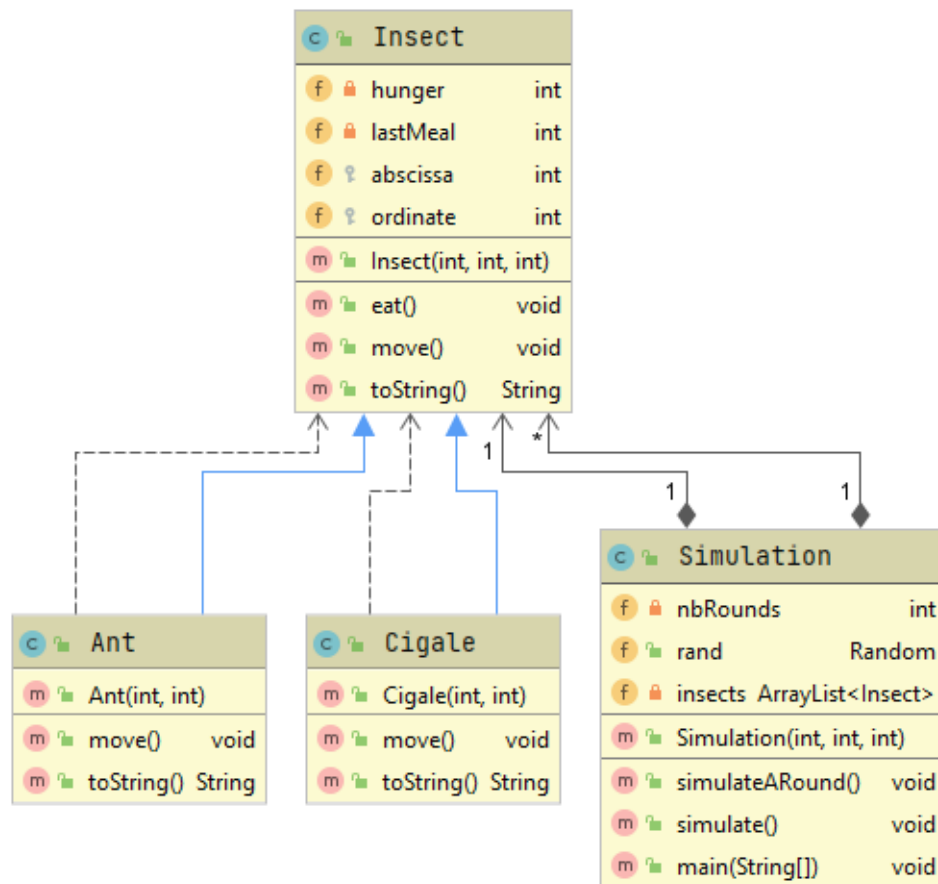
- Redéfinir la méthode **toString** pour qu'elle renvoie le type du triangle : c-à-d « **triangle équilatéral** », « **triangle isocèle** » ou « **triangle scalène** » et ses sommets.
- 4) Écrire la classe **Rectangle** qui hérite de **Polygone**. On supposera que les rectangles issus de cette classe sont à plat (horizontaux/verticaux) :
- écrire un constructeur acceptant le **Point sg** (supérieur gauche) et **id** (inférieur droit)
 - redéfinir la méthode **surface**
 - redéfinir la méthode **toString** pour qu'elle renvoie « rectangle à plat » et ses sommets.
- 5) Écrire la classe **Carre** qui hérite de **Rectangle** (les carrés sont donc eux aussi à plat) :
- écrire un constructeur qui accepte le **Point sg** (supérieur gauche) et la longueur **l** d'un côté
 - redéfinir la méthode **surface**
 - redéfinir la méthode **toString** pour qu'elle renvoie « **carré à plat** » et ses sommets.
- 6) Écrire la classe **ListePolygone**. Celle-ci doit permettre de stocker plusieurs polygones :
- garder le constructeur par défaut (au départ la liste est vide).
 - écrire la méthode **ajoutPolygone** qui ajoute un polygone.
 - écrire la méthode **afficheInfos** qui affiche, pour chaque polygone de la liste : son type, les sommets, son périmètre, sa surface.
 - redéfinir la méthode **toString** pour qu'elle renvoie « liste de N polygones ».

3.3 Simulation d'un mini-écosystème de cigales et fourmis

Objectif pédagogique recherché avec cet exercice : apprendre à utiliser les notions d'abstraction, de classes abstraites et à faire des simulations en Java

L'objectif est de réaliser une hiérarchie de classes permettant de simuler un écosystème très simplifié. Cet écosystème sera uniquement composé de cigales et de fourmis qui devront savoir se déplacer et manger. La simulation s'effectue en mode "tour par tour". À chaque tour, tous les insectes effectueront un déplacement et une action. Chaque insecte mangera au bout d'un nombre déterminé de tours.

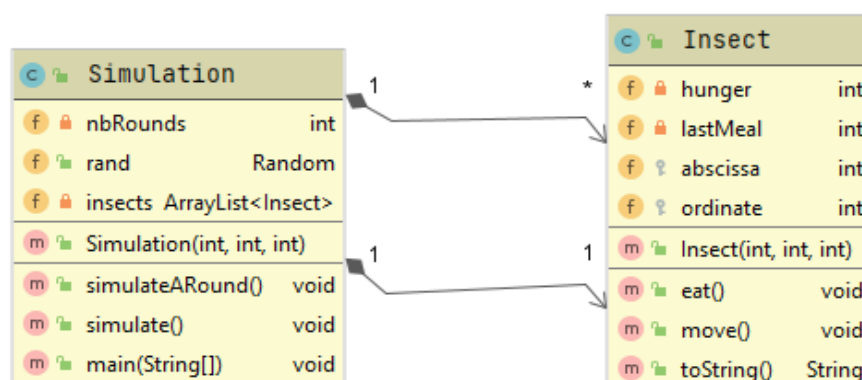
Il y a plusieurs classes à écrire, l'objectif étant d'obtenir par étapes successives un programme correspondant au diagramme ci-dessous :



Travail demandé :

1) Création d'une classe Insect

Dans un premier temps, nous allons créer les classes correspondant au diagramme suivant :



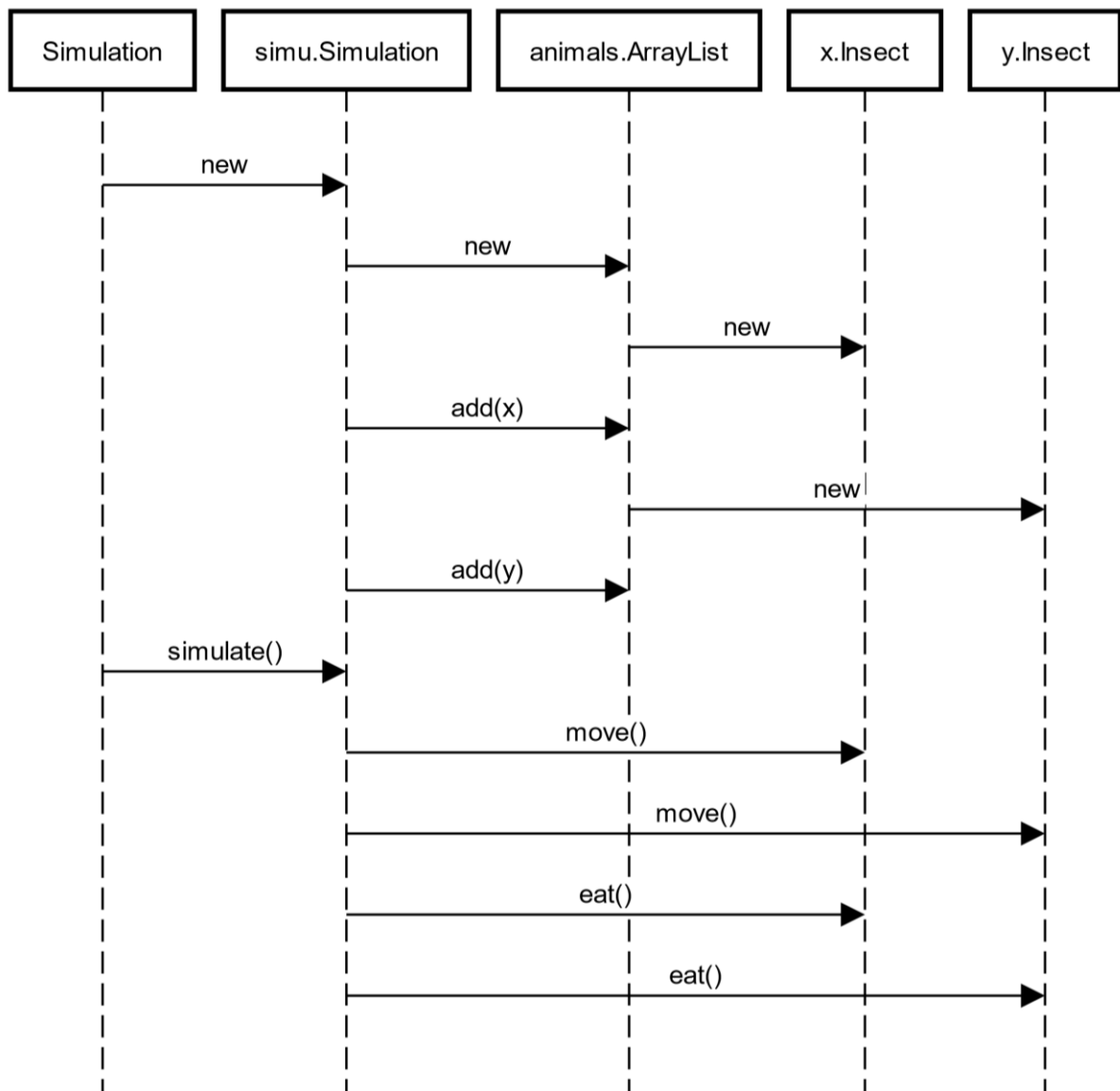
a) Écrire une classe Insect représentant un insecte quelconque.

- i) un insecte possédera une abscisse et une ordonnée, le nombre de tours au bout duquel il a faim, et le nombre de tours depuis son dernier repas ;
- ii) écrire le constructeur de cette classe.

Remarque : souvent, dans vos IDEs vous pouvez générer automatiquement les méthodes get et set d'accès aux variables d'instances. En Eclipse par exemple, utilisez le menu "Source", "Generate Getters and Setters" ou en français "Générer les méthodes d'accès get et set".

- b) Écrire une classe Simulation représentant l'évolution de l'écosystème.
 - i) cette classe possédera une Collection d'insectes (variable de type ArrayList, LinkedList, ou Vector) et le nombre de tours de la simulation ;
 - ii) écrire le constructeur de cette classe. Il prendra comme paramètres le nombre d'insectes et le nombre de tours de la simulation. Il ajoutera le nombre d'insectes nécessaire dans la collection ;
 - iii) écrire une méthode simulate dans cette classe. Pour l'instant, cette méthode se contentera d'afficher tous les insectes ;
 - iv) écrire le main dans la classe Simulation, puis l'exécuter.
- c) Retour sur la classe Insecte :
 - i) écrire la méthode *toString* qui retournera une chaîne de caractères correspondant aux caractéristiques de l'insecte. Réexécuter le programme, et observer la différence avec la première exécution ;
 - ii) écrire la méthode *eat* qui incrémentera le nombre de tours depuis le dernier repas et affichera un message si l'insecte doit manger (nombre de tours maximal atteint). On supposera qu'un insecte trouve tout le temps à manger (donc, s'il a faim il mange).
Remarque : pour correspondre à ce qui est réellement effectué, cette méthode devrait s'appeler *passerUnTourTesterSiTuAsFaimEtManger*, mais pour simplifier son écriture on l'appellera *eat*.
- d) Modification de la classe Simulation :
 - i) dans la méthode simulate, exécuter une boucle de simulation avec le nombre de tours demandé. À chaque tour de boucle, appliquer la méthode *eat* à chaque objet de la collection.

Le fonctionnement du programme obtenu est décrit par le diagramme ci-dessous :

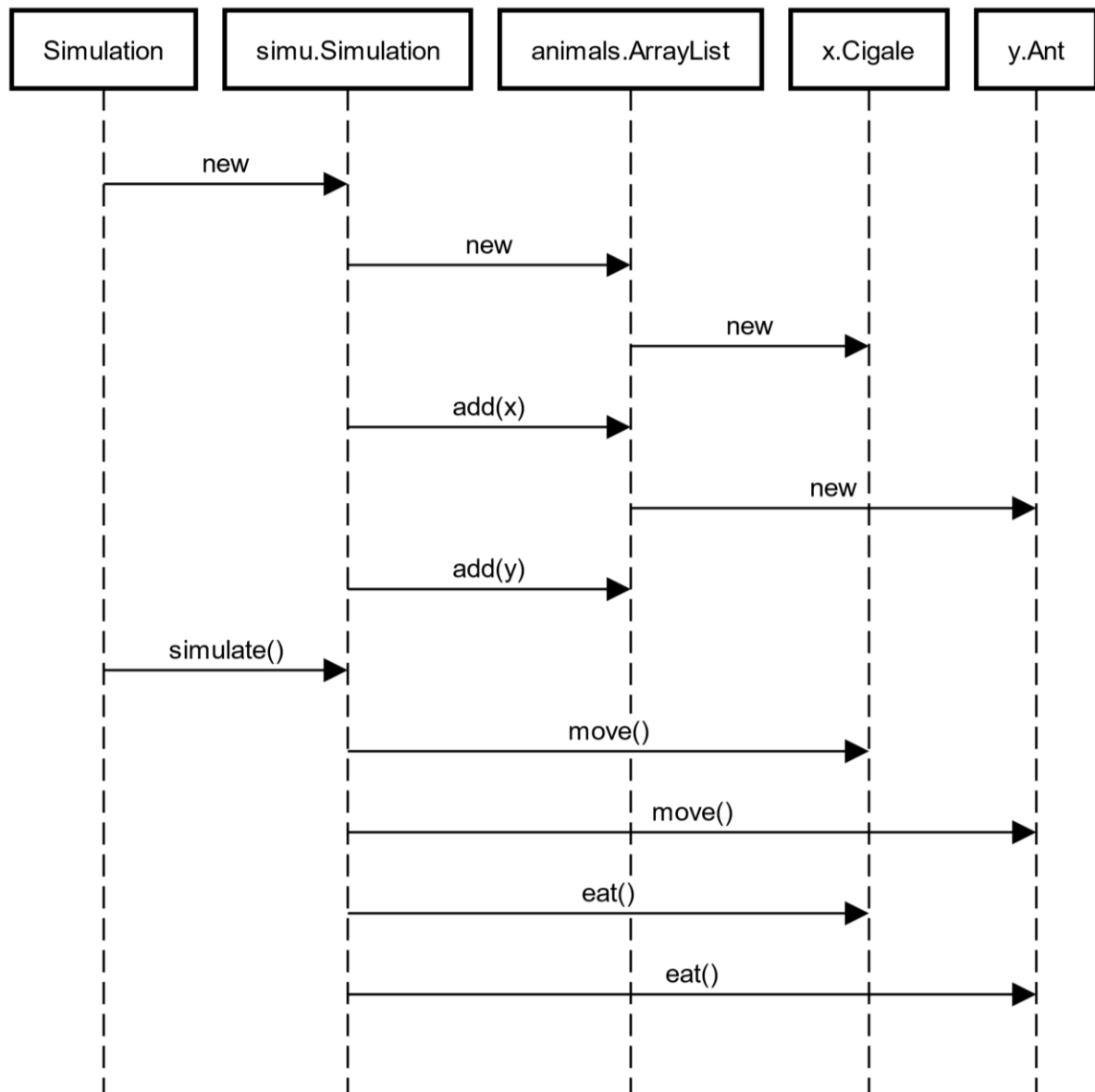


2) Création de sous-classes

Nous allons maintenant ajouter des sous-classes à la classe Insecte pour aboutir au modèle suivant :

- Écrire deux sous-classes de la classe Insecte : Cigale et Ant. Chacune de ces classes possédera un constructeur qui appellera celui d'Insecte, sachant qu'une cigale a faim tous les 3 tours, et une fourmi (ant) tous les 5 tours.
- Écrire la méthode toString des deux sous-classes. Cette méthode retournera le message "I am a cigale, " ou "I am an ant, " suivi du message de la classe Insect.
- Modifier la classe Simulation :
 - ajouter une variable de classe de type Random ;
 - modifier le constructeur pour que les insectes ajoutés à la collection soient aléatoirement des cigales ou des fourmis.
- Exécuter le nouveau programme

Le comportement du nouveau programme est représenté par la figure ci-dessous :

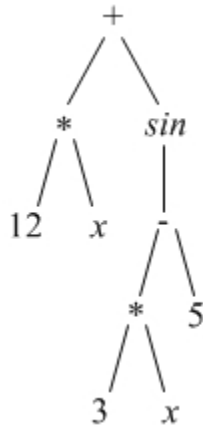


3.4 Expressions arithmétiques (inspiré des TPs de Laurent Tichit)

Objectif pédagogique recherché avec cet exercice : apprendre à utiliser les notions d'abstraction, de classes abstraites et d'interfaces en Java

Dans cet exercice on vous demande de définir un ensemble de classes pour représenter des *fonctions d'une variable* formées avec des constantes, des occurrences de la variable x , les quatre opérations arithmétiques $+$, $-$, \times , $/$ et des appels de quelques fonctions convenues comme *sin*, *cos*, *exp*, *log*, etc. Par exemple :

$12x + \sin(3x - 5)$



Dans un programme, une expression comme celle-là peut être efficacement représentée par une structure arborescente, organisée comme le montre la figure ci-contre, faite de feuilles (les constantes et les variables), de nœuds à deux « descendants » (les opérateurs binaires) et de nœuds à un descendant (les fonctions d'une variable).

Les classes qu'il faut définir sont destinées à représenter les nœuds d'un tel arbre. Il y en a donc de plusieurs sortes :

- un nœud représentant une *constante* porte un nombre, la valeur de la constante,
- un nœud représentant une occurrence de la *variable* x ne porte aucune autre information,
- un nœud représentant une *addition*, une *soustraction*, une *multiplication* ou une *division* porte deux informations : les expressions qui sont ses opérandes,
- un nœud représentant l'appel d'une *fonction* porte comme information l'expressions qui est son argument.

Définissez les classes suivantes (la marge traduit la relation **implements** ou **extends**) :

- 1) **Expression** – *interface* représentant ce qu'ont en commun toutes les expressions arithmétiques (c'est-à-dire toutes les sortes de nœuds de notre structure arborescente) : elle se compose d'une seule méthode :

public double value(double x);

qui renvoie la valeur de l'expression pour la valeur de x donnée. Bien entendu, toutes les classes concrètes de cette hiérarchie devront fournir une définition de la méthode **value**. Elles fourniront aussi une redéfinition intéressante de la méthode **String toString()**.

- 2) **Constant** – classe concrète dont chaque instance représente une occurrence d'une constante. Cette classe a un membre : la valeur de la constante.
- 3) **Variable** – classe concrète dont chaque instance représente une occurrence de la variable x . Cette classe n'a besoin d'aucun membre.
- 4) **OperationBinary** – classe abstraite rassemblant ce qu'ont en commun tous les opérateurs à deux opérandes. Elle a donc deux membres d'instance, de type **Expression**, représentant les deux opérandes, et le constructeur qui va avec.

- 5) **Addition, Soustraction, Multiplication, Division** – classes concrètes pour représenter les opérations binaires. C'est ici qu'on trouve une définition pertinente de la méthode **value** promise dans l'interface **Expression**.
- 6) **OperationUnary** – classe abstraite rassemblant ce qu'ont en commun tous les opérateurs à un opérande. Elle doit avoir un membre d'instance, de type **Expression**, représentant l'opérande en question.
- 7) **Sin, Cos, Log, Exp**, etc. – classes concrètes pour représenter les fonctions standard. Ici on doit trouver une définition pertinente de la méthode **value** promise dans l'interface **Expression**.

On ne vous demande pas de résoudre le problème (difficile) de la « lecture » d'un tel arbre, c'est-à-dire de sa construction à partir d'un texte, par exemple lu à la console. En revanche, vous devez montrer que votre structure est bien adaptée au calcul de la valeur de l'expression pour une valeur donnée de la variable *x*. Pour cela, on exécutera un programme d'essai comme celui-ci :

```
...
public static void main(String[] args) {
    /* coding of the function f(x) = 2 * sin(x) + 3 * cos(x) */
    Expression f = new Addition(
        new Multiplication(
            new Constant(2), new Sin(new Variable()),
            new Multiplication(
                new Constant(3), new Cos(new Variable())));
    /* calculate the value of f(x) for some values of x */
    double[] tab = { 0, 0.5, 1, 1.5, 2, 2.5 };
    for (int i = 0; i < tab.length; i++) {
        double x = tab[i];
        System.out.println("f(" + x + ") = " + f.valeur(x));
    }
}
...
```

L'exécution de ce programme produit l'affichage :

f(0.0) = 3.0

f(0.5) = 3.5915987628795243

f(1.0) = 3.3038488872202123

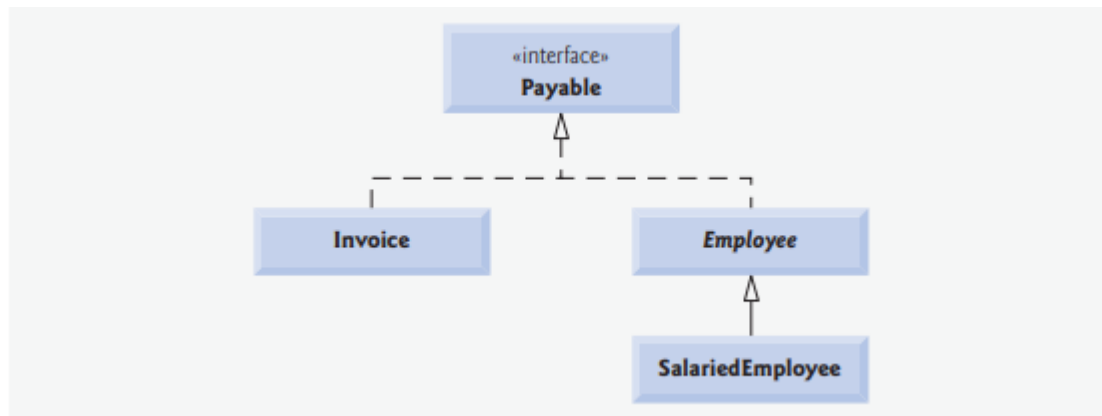
f(1.5) = 2.2072015782112175

f(2.0) = 0.5701543440099361

f(2.5) = -1.2064865584328883

3.5 Développer une hiérarchie des comptes créditeurs

Objectif pédagogique recherché avec cet exercice : apprendre à utiliser l'interfaces



Travail demandé :

- 1) Créez l'interface Payable, qui contient la méthode `getPaymentAmount()` qui renvoie un montant double qui doit être payé pour un objet de toute classe qui implémente l'interface.
- 2) Créez la classe Invoice pour représenter une facture simple qui contient des informations de facturation pour un seul type de pièce. La classe déclare des variables d'instance privées `partNumber`, `partDescription`, `quantity` et `pricePerItem` qui indiquent le numéro de la pièce, une description de la pièce, la quantité de la pièce commandée et le prix par article.
- 3) Créez la classe Employee de manière à ce qu'elle implémente l'interface Payable. La classe déclare des variables d'instance privées `firstName`, `lastName` et `socialSecurityNumber`
- 4) Créez la classe SalariedEmployee qui étend Employee et remplit le contrat de la superclasse Employee pour implémenter la méthode `getPaymentAmount` de l'interface Payable. La classe déclare la variable d'instance privée `weeklySalary`.

4 Gestion d'erreurs

4.1 Exceptions (© Dr Robert Harle)

Objectif pédagogique recherché avec cet exercice : apprendre à lancer des exceptions

```
public class RetValTest {
    public static String sEmail = "";
    public static int extractCamEmail(String sentence) {
        if (sentence==null || sentence.length()==0)
            return -1; // Error - sentence empty
        String tokens[] = sentence.split(" "); // split into tokens
        for (int i=0; i< tokens.length; i++) {
            if (tokens[i].endsWith("@cam.ac.uk")) {
                sEmail=tokens[i];
                return 0; // success
            }
        }
        return -2; // Error - no cam email found
    }
    public static void main(String[] args) {
        int ret=RetValTest.extractCamEmail("My email is rkh23@cam.ac.uk");
```

```

        if (ret==0) System.out.println("Success: "+RetValTest.sEmail);
        else if (ret==-1) System.out.println("Supplied string empty");
        else System.out.println("No @cam address in supplied string");
    }
}

```

Travail demandé :

- 1) Ce code permet de saisir les erreurs à l'aide des valeurs de retour. Réécrivez-le pour utiliser des exceptions de type `RuntimeException` à la place des retours -1 et -2.

4.2 Lecture d'un fichier

Objectif pédagogique recherché avec cet exercice : apprendre à lancer des exceptions

Pour lire à partir d'un fichier vous aurez toujours besoin d'utiliser une déclaration try/catch en Java.

Le code de base suivant est nécessaire pour ouvrir un fichier.

```

import java.util.Scanner;
import java.io.*;
public class ReadFile {
    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter the name of the file: ");
        String filename = in.next();
        BufferedReader bf = new BufferedReader(new FileReader(filename));
        String line = bf.readLine();
        while (line != null) {
            System.out.println(line);
            line = bf.readLine();
        }
    }
}

```

Travail demandé :

Si le fichier spécifié n'existe pas, il lance une `FileNotFoundException`. Réécrivez ce code pour gérer l'exception. Si le fichier n'existe pas, continuez à demander un nouveau fichier à l'utilisateur.

4.3 Les numéros (© Chua Hock-Chuan)

Objectif pédagogique recherché avec cet exercice : Tester les classes Java par l'exemple

Supposons que nous ayons une classe appelée `MyNumber` qui représente un nombre, et capable d'effectuer des opérations arithmétiques.

Travail demandé

- 1) Créer un projet Java appelé "JUnitTest"
- 2) Créer une nouvelle classe Java appelée "MyNumber", comme suit :

```

public class MyNumber {
    /**
     * The class MyNumber represent a number, and capable
     * of performing arithmetic operations.
     */

    int number;

    // Constructor
    public MyNumber() {
        this.number = 0;
    }

    public MyNumber(int number) {
        this.number = number;
    }

    // Getter and setter
    public int getNumber() {
        return number;
    }

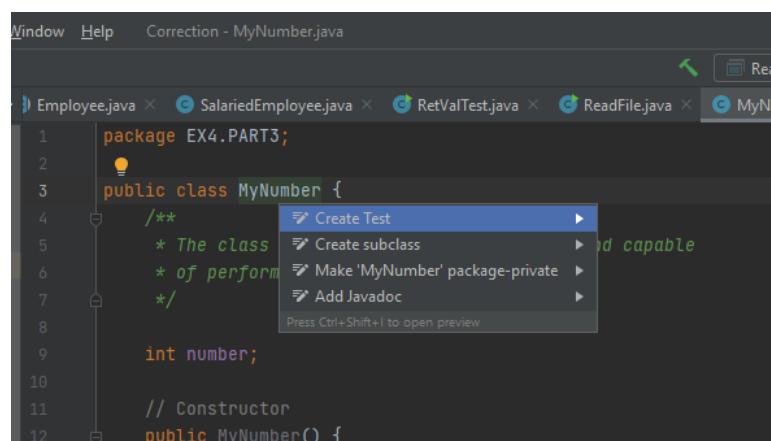
    public void setNumber(int number) {
        this.number = number;
    }

    // Public methods
    public MyNumber add(MyNumber rhs) {
        this.number += rhs.number;
        return this;
    }

    public MyNumber div(MyNumber rhs) {
        if (rhs.number == 0) throw new IllegalArgumentException("Cannot
divide by 0!");
        this.number /= rhs.number;
        return this;
    }
}

```

3) Créez un nouveau test en cliquant sur le nom de la classe et ensuite Alt+Enter.



4) Créez le premier cas de test appelé MyNumberTest , qui fait comme suit :

- Définit deux nombres (Constructeur) avant les tests
- Teste le getter et le setter

- Teste la méthode d'ajout
 - Teste la méthode de division
 - Teste la division par zéro
- 5) Exécutez le test et observez le résultat. Modifiez certaines lignes pour faire échouer le test et observez le résultat.

4.4 Compte bancaire

Objectif pédagogique recherché avec cet exercice : mettre en pratique les tests unitaires java

A partir du "cahier des charges" de l'exercice Compte bancaire présenté auparavant, élaborer une spécification d'une classe Java modélisant un compte bancaire.

Travail demandé

1. Écrire un programme de test simple permettant de :
 - créer un compte c1, au nom de J. DUPONT avec un solde initial de 1 000 €
 - créer un compte c2, au nom de C. DURANT avec un solde initial de 50 000 €, un débit maximal autorisé de 6000 € et un découvert maximal autorisé de 5000 €
 - d'afficher les caractéristiques des comptes c1 et c2 (c'est à dire les informations suivantes : numéro du compte, nom du titulaire, découvert maximal autorisé, débit maximal autorisé, solde du compte et si le compte est à découvert un message le signalant explicitement)
 - retirer 300 € du compte c1
 - retirer 600 € du compte c2
 - déposer 500 € sur le compte c1
 - d'afficher les caractéristiques des comptes c1 et c2
 - virer 1000 € du compte c2 vers le compte c1
 - d'afficher les caractéristiques des comptes c1 et c2.
2. Ajouter d'autres tests vous paraissant pertinents

4.5 Factorials

Objectif pédagogique recherché avec cet exercice : Throwing Exceptions

Le fichier Factorials.java contient un programme qui appelle la méthode factorielle de la classe MathUtils pour calculer les factoriels des entiers saisis par l'utilisateur. Enregistrez ces fichiers dans votre répertoire et étudiez le code des deux fichiers, puis compilez et exécutez Factorials pour voir comment cela fonctionne. Essayez plusieurs entiers positifs, puis essayez un nombre négatif. Vous devriez constater que cela fonctionne pour les petits nombres positifs (valeurs < 17), mais qu'il renvoie une grande valeur négative pour les grands nombres et qu'il renvoie toujours 1 pour les entiers.

```
// *****
// Factorials.java
//
// Reads integers from the user and prints the factorial of each.
//
```

```
// *****
import java.util.Scanner;
public class Factorials
{
    public static void main(String[] args)
    {
        String keepGoing = "y";
        Scanner scan = new Scanner(System.in);
        while (keepGoing.equals("y") || keepGoing.equals("Y"))
        {
            System.out.print("Enter an integer: ");
            int val = scan.nextInt();
            System.out.println("Factorial(" + val + ") = "
                               + MathUtils.factorial(val));
            System.out.print("Another factorial? (y/n) ");
            keepGoing = scan.next();
        }
    }
}
```

```
// *****
// MathUtils.java
//
// Provides static mathematical utility functions.
//
// *****
public class MathUtils
{
    //-----
    // Returns the factorial of the argument given
    //-----
    public static int factorial(int n)
    {
        int fac = 1;
        for (int i=n; i>0; i--)
            fac *= i;
        return fac;
    }
}
```

Retourner 1 quand la factorielle de tout entier négatif n'est pas correct - mathématiquement, la fonction factorielle n'est pas définie pour les entiers négatifs. Pour corriger cela, vous pourriez modifier votre méthode factorielle pour vérifier si l'argument est négatif, mais alors quoi ? La méthode doit renvoyer une valeur, et même si elle affiche un message d'erreur, la valeur renvoyée, quelle qu'elle soit, pourrait être mal interprétée. Elle devrait plutôt lancer une exception indiquant que quelque chose s'est mal passé, de sorte qu'elle ne pourrait pas terminer son calcul. Vous pouvez définir votre propre classe d'exception, mais il existe déjà une exception adaptée à cette situation : `IllegalArgumentException`, qui étend `RuntimeException`. Modifiez votre programme comme suit :

- 1) Modifiez l'en-tête de la méthode factorielle pour indiquer que la méthode factorielle peut lancer une `IllegalArgumentException`.
- 2) Modifier le corps de la factorielle pour vérifier la valeur de l'argument et, s'il est négatif, lancer une `IllegalArgumentException`. Notez que ce que vous passez pour lancer est en fait une instance de la `IllegalArgumentException` class, et que le constructeur prend un paramètre `String`. Utilisez ce paramètre pour être précis sur la nature du problème.
- 3) Après avoir effectué ces changements, compilez et exécutez votre programme `Factorials`. Maintenant, lorsque vous entrez un nombre négatif, une exception sera lancée, mettant fin au programme. Le programme se termine parce que l'exception n'est pas prise en compte, elle est donc lancée par la méthode principale, ce qui provoque une erreur d'exécution.

- 4) Modifiez la méthode principale dans votre classe de Factorials pour attraper l'exception lancée par la factorielle et imprimez un message approprié, mais continuez ensuite avec la boucle. Réfléchissez bien à l'endroit où vous devrez mettre les try et catch.

5 JavaFX

5.1 Interface Graphique Simple

Objectif pédagogique recherché avec cet exercice : Apprendre à créer une interface simple

Travail demandé :

Écrivez un programme qui affiche quatre lignes de texte dans quatre étiquettes :

- 1) Écrivez votre nom, votre ID, votre adresse, votre téléphone dans les étiquettes respectivement.
- 2) Mettez le fond des étiquettes en blanc.
- 3) Réglez la couleur du texte des étiquettes sur noir, bleu, cyan, vert, respectivement.
- 4) Définissez la police de chaque étiquette sur Times New Roman, en gras, et 20 pixels.
- 5) Définissez la bordure de chaque étiquette comme une bordure de ligne de couleur jaune.

5.2 Simple Transitions (The Australian National University)

Objectif pédagogique recherché avec cet exercice : Apprendre à utiliser les actions, les figures et les couleurs

Créer un programme, appelé SimpleTransitions.java. Qui affiche une forme simple avec de la couleur, puis effectue l'une des opérations suivantes lorsqu'un événement est généré :

- 1) se déplace vers un autre endroit (choisissez vous-même la direction et la distance) et y reste lorsque vous cliquez dessus
- 2) tourne de 45° dans le sens des aiguilles d'une montre lorsque vous appuyez sur la touche "UP", et de 45° dans le sens inverse lorsque vous appuyez sur "DOWN".

5.3 Voitures de course

Objectif pédagogique recherché avec cet exercice : Apprendre à utiliser les actions, les images ...

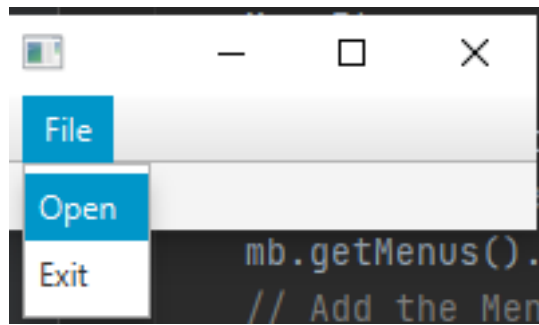
Travail demandé :

Écrivez un programme qui anime le mouvement des voitures de course sur une piste de course de forme ovale. Utilisez les boutons Stop et Resume pour contrôler le mouvement de la voiture. Trouvez une image de voiture à utiliser dans ce programme.

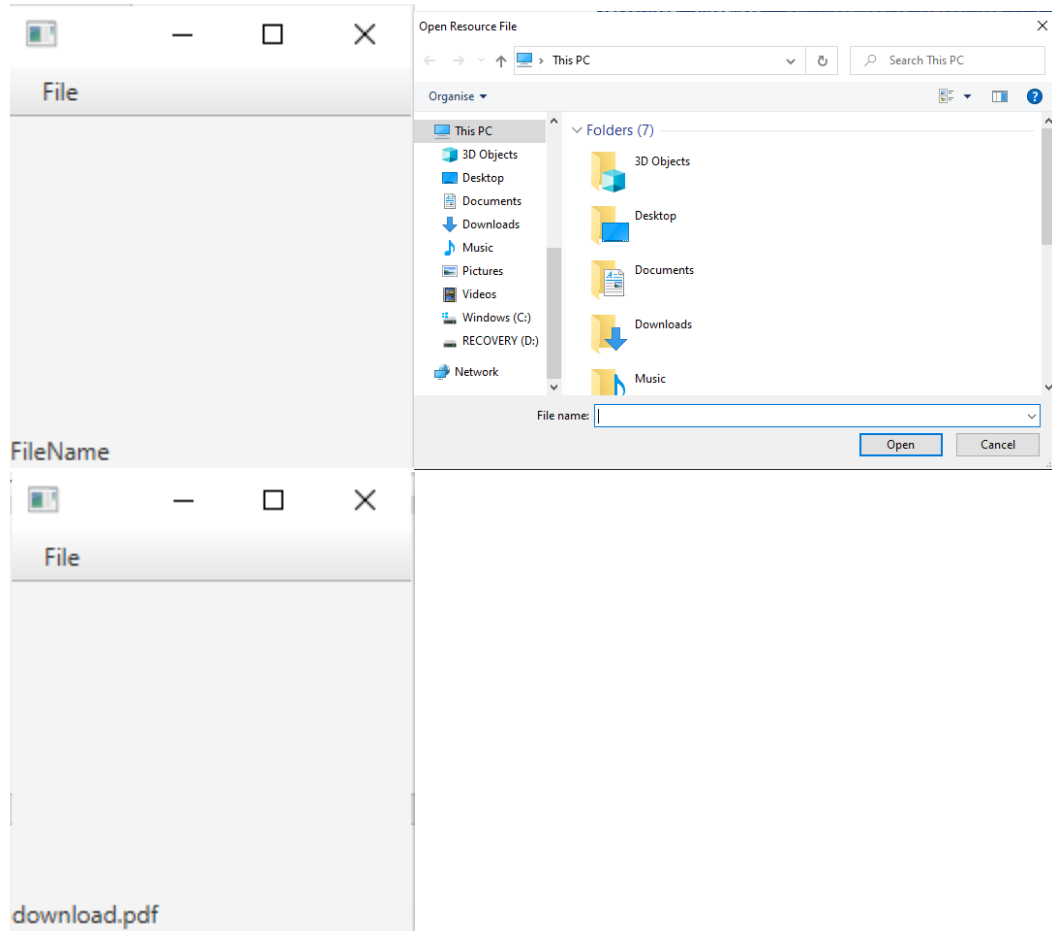
5.4 Menus (PARIS DIDEROT)

Objectif pédagogique recherché avec cet exercice : Apprendre à utiliser les menus de JavaFX

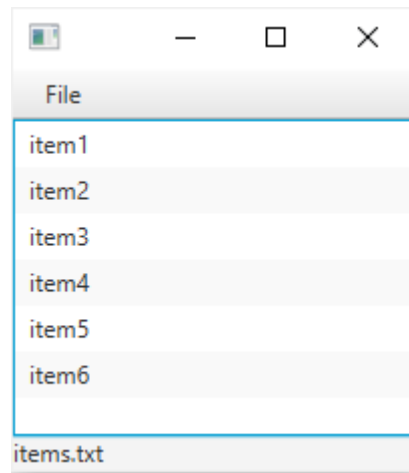
- 1) Créer une application JavaFX qui ouvre une fenêtre qui contient un menu avec deux choix : Open et Exit. La sélection du second choix devra terminer l'application. Le menu sera placé au sommet d'un BorderPane.
- 2) Modifier l'application de telle sorte que la terminaison puisse être obtenue via un raccourci clavier (Ctrl-X, par exemple).



- 3) Reprendre le programme. Ajouter un label en bas du BorderLayout qui aura pour valeur (au lancement) Filename.
- 4) Faire en sorte qu'en sélectionnant l'option Open du menu, un panneau de sélection de fichier s'affiche et qu'en choisissant un fichier son nom s'affiche dans le label.



- 5) Ajouter des filtres au panneau de sélection de sorte que l'utilisateur puisse choisir entre Text files d'extension *.txt et All files d'extension *.*.
- 6) Ajouter dans la zone centrale du BorderLayout un ListView. Faire en sorte que lorsqu'un fichier est sélectionné, chaque ligne du fichier (.lst) est insérée dans la liste centrale.



5.5 SliderDemo

Objectif pédagogique recherché avec cet exercice : Apprendre à utiliser les sliders de JavaFX

Un curseur nous permet de contrôler la valeur d'une variable en déplaçant une barre coulissante, qui est utilisée pour faire varier la valeur dans une plage particulière. L'application que nous avons développée dans SliderDemo présente deux curseurs, un horizontal et un vertical. La valeur actuelle du "pouce" mobile sur chaque curseur est affichée sous le curseur.

```
import java.text.DecimalFormat;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

public class SliderDemo extends Application {
    @Override
    public void start(Stage stage) {
        final double horizSliderWidth = 300;
        final double vertSliderHeight = 300;
        // numbers will be formatted to one decimal place
        DecimalFormat df = new DecimalFormat("0.0");
        // create and configure the vertical slider
        Slider vertSlider = new Slider(0, 20, 0);
        vertSlider.setMinHeight(vertSliderHeight);
        vertSlider.setShowTickMarks(true);
        vertSlider.setShowTickLabels(true);
        vertSlider.setSnapToTicks(true);
```

```

        vertSlider.setMajorTickUnit(5.0);
        vertSlider.setMinorTickCount(10);
        vertSlider.setOrientation(Orientation.VERTICAL); // default is
horizontal
// create and configure the horizontal slider
        Slider horizSlider = new Slider(0, 10, 0);
        horizSlider.setMinWidth(horizSliderWidth);
        horizSlider.setShowTickMarks(true);
        horizSlider.setShowTickLabels(true);
        horizSlider.setSnapToTicks(true);
        horizSlider.setMajorTickUnit(1.0);
        horizSlider.setMinorTickCount(4);
// create two labels to keep track of each slider position
        Label horizLabel = new Label("Current value is 0.0");
        Label vertLabel = new Label("Current value is 0.0");
// add a listener to the vertical slider
        vertSlider.valueProperty().addListener((observable, oldValue,
newValue) ->
                vertLabel.setText("Current value is " +
df.format(newValue)));
// add a listener to the horizontal slider
        horizSlider.valueProperty().addListener((obsValue, oldValue,
newValue) ->
                horizLabel.setText("Current value is " +
df.format(newValue)));
// create and configure a VBox to hold the vertical slider and label
        VBox vertBox = new VBox(10);
        vertBox.setAlignment(Pos.BOTTOM_LEFT);
        vertBox.setMinWidth(horizSliderWidth / 3);
        vertBox.getChildren().addAll(vertSlider, vertLabel);
// create and configure a VBox to hold the horizontal slider and label
        VBox horizBox = new VBox(10);
        horizBox.setAlignment(Pos.BOTTOM_LEFT);
        horizBox.getChildren().addAll(horizSlider, horizLabel);
// create and configure an HBox as root
        HBox root = new HBox(30);
        root.setPadding(new Insets(10, 10, 10, 10));
        root.getChildren().addAll(horizBox, vertBox);
// create and configure the scene and stage
        Scene scene = new Scene(root, 460, 350);
        stage.setScene(scene);
        stage.setTitle("Slider Example");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

Travail demandé

- 1) Adaptez le SliderDemo de sorte qu'au lieu d'imprimer une valeur entière, il dessine un rectangle qui s'agrandit à mesure que le curseur est déplacé. Ceci est démontré dans le diagramme suivant :

