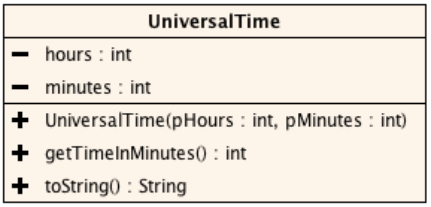
Série B : Interfaces graphiques

Table des matières

Exercice B.1: 'Universal Time' avec interface

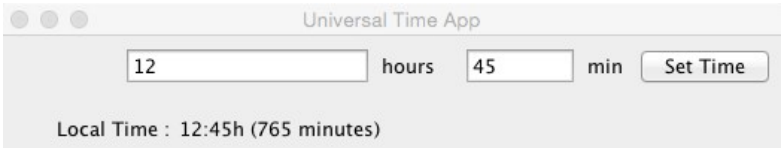


1. Développez une classe **`UniversalTime`** avec un constructeur à deux paramètres pour initialiser les deux attributs **`hours`** et **`minutes`**. Ajoutez une méthode **`getTimeInMinutes`** qui retourne l'heure exacte en minutes (P.ex. pour 12:20h la méthode retourne 740). Ajoutez à cette classe une méthode **`toString`** qui retourne l'heure en notation normale ainsi que l'heure en minutes. P.ex. pour 12:20h la méthode **`toString`** retourne :

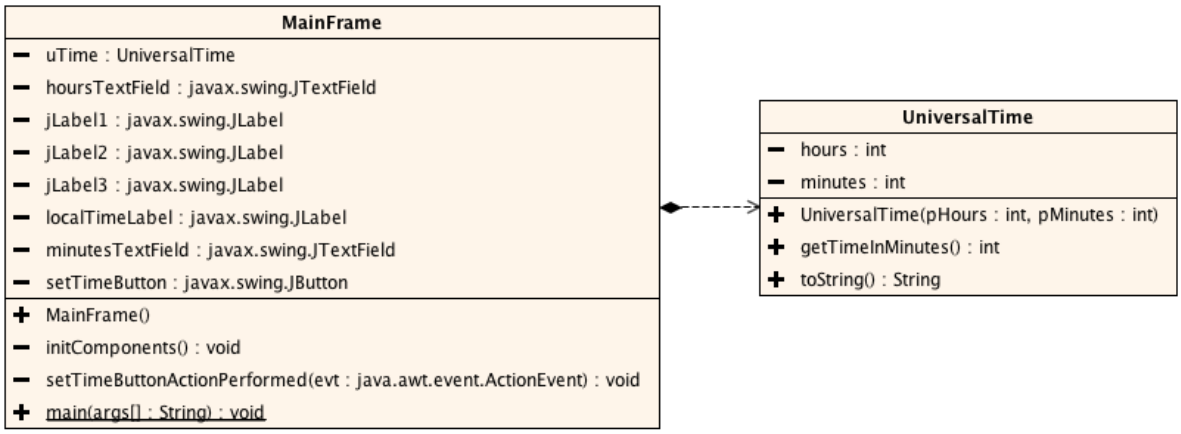
   `12:20h (740 minutes)`

2. *Interface texte avec Unimozer :*
   Développez une deuxième classe **`UniversalTimeController`** avec une méthode **`run`** qui crée automatiquement une instance **`uTime`** de la classe **`UniversalTime`** en l'initialisant avec une heure de votre choix et qui affiche la valeur retournée de la méthode **`toString`** dans la fenêtre des messages.

3. *Interface graphique avec NetBeans :*



   Sauvegardez le projet et fermez Unimozer. Ouvrez le projet en Netbeans et ajoutez une nouvelle fiche **`MainFrame`**. Au début de la classe **`MainFrame`** déclarez et initialisez l'instance **`uTime`** du type **`UniversalTime`**. Ajoutez un bouton **`setTimeButton`**, deux champs d'édition (**`hoursTextField`**, **`minutesTextField`**) et un libellé **`localTimeLabel`**. Des libellés supplémentaires sont ajoutés 'comme décoration' pour que l'utilisateur reconnaisse mieux les éléments sur la fiche. Lors d'un clic sur le bouton **`setTimeButton`**, **`uTime`** est réinitialisé par le temps indiqué dans les champs d'édition. Le temps est affiché dans **`localTimeLabel`** en faisant appel à **`toString`** de **`uTime`**.



2GIN :

4. *Application texte avec Unimozer :*
   Reprenez la classe originale en Unimozer. Développez une deuxième classe **`Main`** et cochez cette fois la case **[X] Main Method**. La classe **`Main`** contiendra alors une méthode **`main`** qui se présente comme suit :
   ```
   public static void main(String[] args)
   ```
   Indiquez dans le bloc d'instructions de cette méthode **`main`** les mêmes instructions que dans la méthode **`run`** du point 2.
   Grâce à la présence de la méthode **`main`**, vous pouvez maintenant démarrer le programme (→ Project → Run [F6]) sans devoir créer une instance de la classe. (On vous demandera une liste d'arguments pour le programme, mais vous pouvez confirmer en laissant la liste vide).

5. *Application texte avec Netbeans :*
   Créez une nouvelle application **`B01_5_UniversalTimeController_NetBeans`** dans Netbeans en cochant ***exceptionnellement pour cet exercice*** la case **[X] Create Main Class.** Importez la classe **`UniversalTime`** dans cette application. Dans la méthode **`main`** de la classe **`Main`**, vous pouvez entrer les mêmes instructions que dans la classe **`main`** du point précédent. Vous pouvez lancer l'application avec **Run** ou  **.**

Notions requises : Création de nouveaux objets, importation de projets Unimozer dans Netbeans, création de projets dans Netbeans.

Exercice B.2: 'Universal Time' version finale

Partie 1:

Copiez le projet de l'exercice précédent et sauvegardez-le sous **`B02_UniversalTime`** (p.ex en NetBeans en effectuant un clic droit sur le projet et en choisissant copy...).

Complétez la classe **`UniversalTime`** comme suit (consultez le modèle UML de la page suivante):

- Définissez une méthode privée[1] **`valueOf`** qui prend comme paramètres **`pHours`** et **`pMinutes`** et qui retourne le temps donné par ces paramètres sous le format **`<heure>:<minutes>h`** P.ex : **`12:50h`**
- Définissez la méthode **`getLocalTime`** qui retourne le temps actuel en faisant appel à **`valueOf`**.
- Définissez la méthode **`setToSystemTime`**() qui initialise les attributs **hours** et **minutes** avec le temps local actuel.

  Ajoutez comme première ligne de votre classe **`UniversalTime`** la ligne suivante :

```
1
2  import java.time.LocalTime;
3
4  public class UniversalTime
5  {
```

  Voici le code de la méthode:

```
public void setToSystemTime()
{
   //Créer une nouvelle instance now de la classe prédéfinie LocalTime
   LocalTime now = LocalTime.now();
   // Utiliser l'instance now pour obtenir l'heure et les minutes actuelles
   hours   = now.getHour();
   minutes = now.getMinute();
}
```

  Dans la classe **`MainFrame`**, ajoutez un bouton **`systemTimeButton`** qui utilise cette méthode et qui affiche le temps système (de l'ordinateur) dans le libellé **localTimeLabel**.

1 La méthode **valueOf** est privée puisque sa fonctionnalité n'est pas utile à l'extérieur de la classe. Il s'agit plutôt d'une méthode d'aide qui facilite la programmation des méthodes **get...Time** à l'intérieur de la classe. En POO on essaie par principe de publier (rendre accessibles vers l'extérieur) uniquement les éléments vraiment nécessaires à l'utilisation de la classe.

- Ajoutez ensuite les méthodes suivantes (en profitant de **`valueOf`**) :

  `getUtcTime` qui retourne le temps du fuseau horaire

  `getNewYorkTime` qui retourne le temps à New York.

  `getTallinnTime` qui retourne le temps à Tallinn.

  `getLosAngelesTime` qui retourne le temps à Los Angeles.

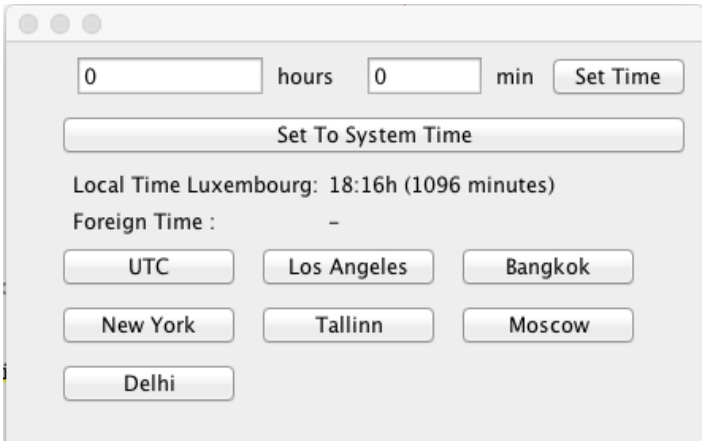
  `getBangkokTime` qui retourne le temps à Bangkok.

  `getMoscowTime` qui retourne le temps à Moscou.

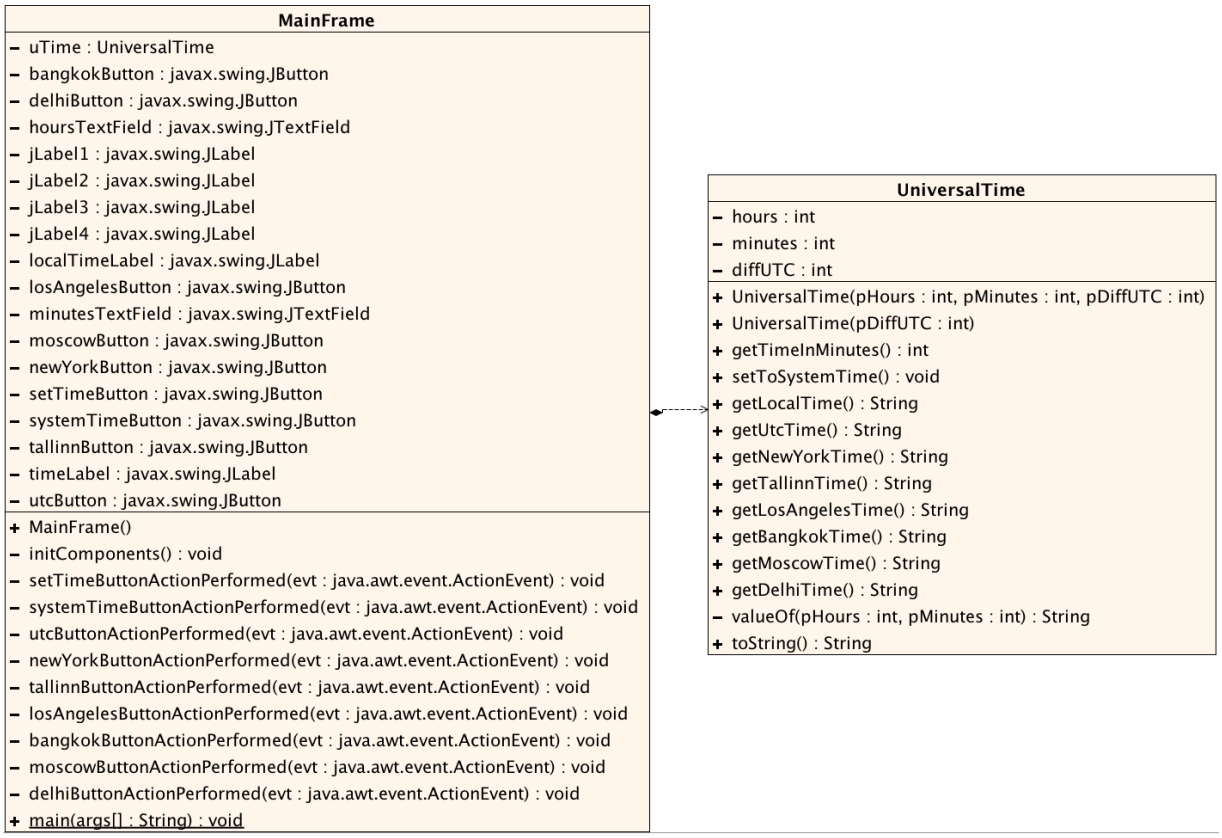
  `getDelhiTime` qui retourne le temps à Delhi.

  Recherchez les différences UTC sur Internet. Evitez de retourner des temps négatifs.

  Dans la classe **`MainFrame`**, ajoutez des boutons qui font appel aux différentes méthodes. Les résultats sont affichés dans un second libellé **`foreignTimeLabel`**.



Voici le modèle UML :



Partie 2:

Créez une copie du projet.

Ajoutez maintenant à la classe un attribut **`diffUTC`** qui contient la différence entre le temps local et le temps GMT/UTC (valeur : +1 pour Luxembourg). Cette différence doit aussi être indiquée dans le constructeur (on ne tient pas compte de l'heure d'été).

Modifiez les calculs pour utiliser cet attribut pour déterminer le temps UTC à partir du temps actuel. Le temps UTC sera alors utilisé pour calculer le temps dans les autres zones.

Le calcul se fait donc comme suit:

temps local -> temps UTC -> temps autre zone

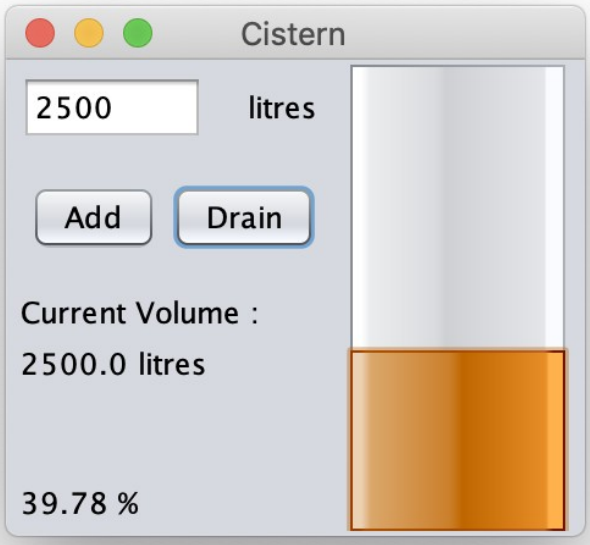
Pour avancés / 2GIN :

Ajoutez un troisième constructeur qui initialise la classe automatiquement par le temps local actuel.

Ajoutez à **`UniversalTime`** un attribut **`daylightSavingTime`** que vous pouvez activer en été. Dans ce cas, la différence du Luxembourg par rapport au temps UTC est de 2 heures. Vous devez adapter aussi les méthodes de différentes villes.

Notions requises :	Création de nouveaux objets, importation de projets Unimozer dans Netbeans, création de projets dans Netbeans
Composants requis:	JButton, JLabel

Exercice B.3: Citerne



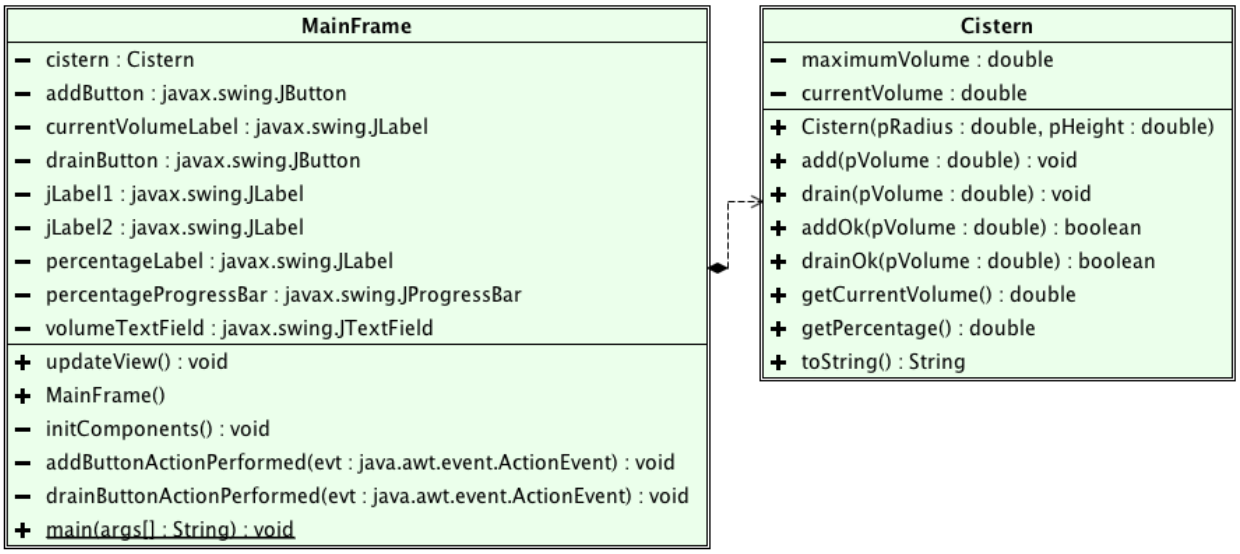
En vous basant sur l'exercice réalisé dans le cours de 3GIG (classe **`Cistern`**), développez un programme simulant une citerne d'eau de rayon 1 m et ayant 2 m en hauteur. L'utilisateur peut à l'aide de l'interface utilisateur

- ajouter de l'eau à la citerne et
- retirer de l'eau de la citerne.

A tout moment, le taux de remplissage (en litres et en pourcentage) de la citerne doit être affiché à l'écran.

Définissez une méthode **`updateView`** pour l'affichage de l'état actuel de la citerne sur la fiche. Profitez de cette méthode lors de chaque modification de la citerne.

Modèle UML :



Remarque MVC :

En fait, la méthode **`updateView`** 'synchronise' la vue (**`MainFrame`**) avec le modèle (**`Cistern`**). Dans la plupart de nos applications une telle méthode sera très utile.

Améliorations à réaliser :

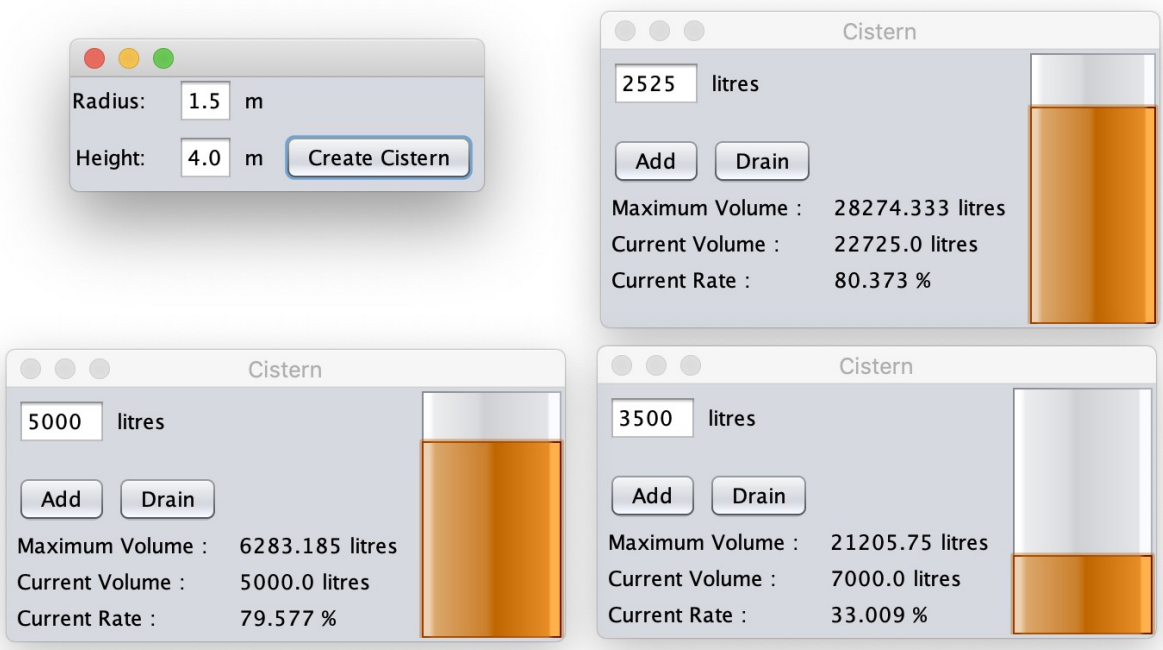
- Ajoutez une barre de progrès (**`ProgressBar`**) pour montrer le taux de remplissage.
- Trouvez un truc (→ calcul) pour limiter le nombre de positions décimales du taux de remplissage à 2.

Question :

En réalisant ces améliorations, vous avez certainement profité de la méthode **`updateView`**. Plus généralement, quels sont les avantages d'une telle méthode lors du développement d'un programme ?

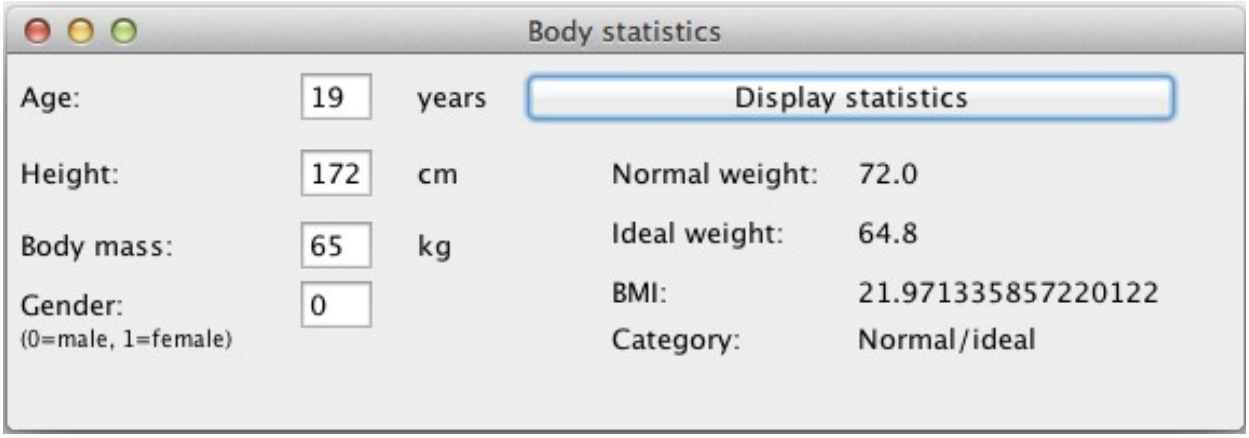
Pour avancés / 2GIN :

En profitant au mieux du travail déjà réalisé, modifiez le programme pour pouvoir afficher plusieurs citernes de différentes tailles? Par exemple :

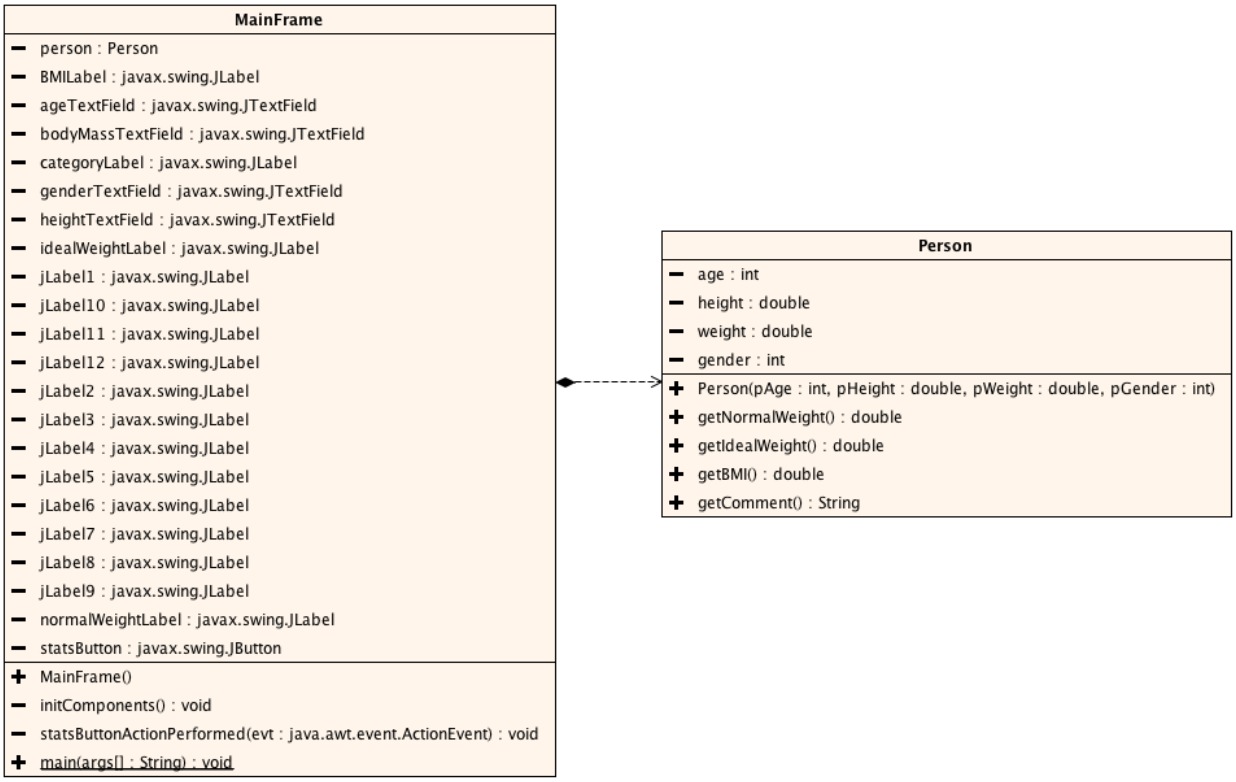


Notions requises: MVC, new, typecast
Composants requis: JFrame, JTextField, JButton, JLabel, JPanel

Exercice B.4: Statistiques corporelles



En vous basant sur l'exercice réalisé dans le cours de 3GIG (classe **`Person`**), développez un programme qui saisit les données d'une personne (âge, taille, sexe, poids) et qui affiche ensuite le poids normal et le poids idéal selon la formule de Broca, le BMI ainsi qu'une appréciation de la situation pondérale de la personne.

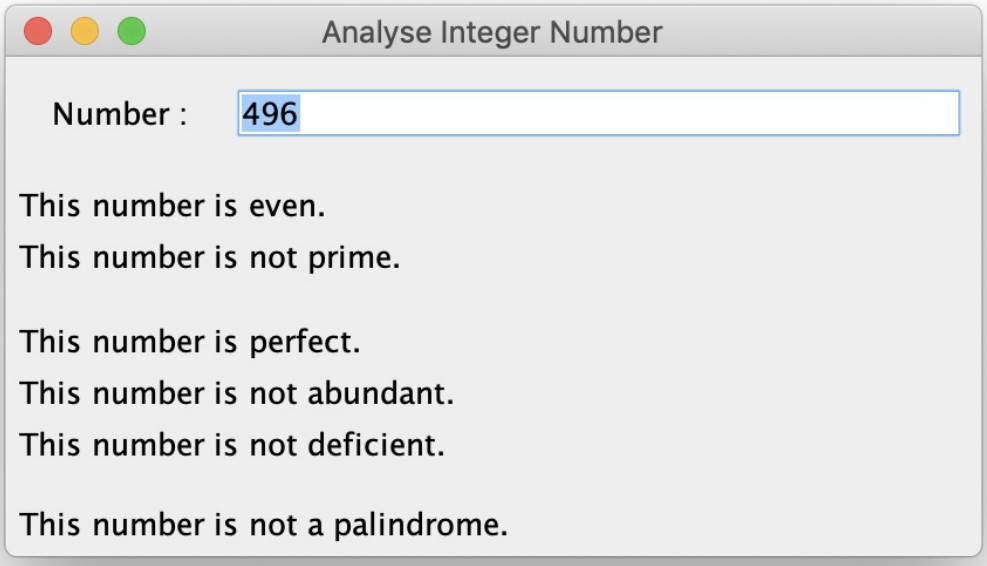


2GIN :

Utilisez des composants **`RadioButton`** pour saisir le sexe et groupez-les à l'aide d'un **`ButtonGroup`**.

Notions requises: MVC, new, typecast
Composants requis: JFrame, JTextField, JButton, JLabel

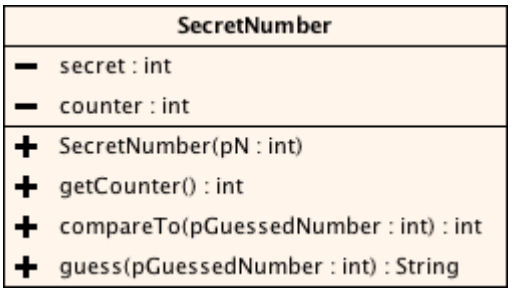
Exercice B.5: Analyse d'un entier



En vous basant sur l'exercice réalisé dans le cours de 3GIG (classe **`IntegerNumber`**), développez un programme qui saisit un nombre entier et qui affiche ensuite les caractéristiques de ce nombre (pair, premier, parfait, déficient, abondant, palindrome).

Notions requises: MVC, new, typecast
Composants requis: JFrame, JTextField, JLabel

Exercice B.6: Secret Number (Devinette)



En vous basant sur l'exercice réalisé dans le cours de 3GIG (classe **`NumberPuzzle`**), développez un programme où:

- le constructeur **`SecretNumber`** choisit au hasard un nombre entre 1 et pN.
- **`compareTo`** retourne -1, 0 ou 1 si le nombre donné est plus petit, égal ou plus grand que le nombre secret.
- **`getCounter`** retourne le nombre d'essais.
- Réalisez un interface graphique pour le jeu. Dans notre jeu la limite supérieure sera fixée à 100. Après que le joueur a deviné le nombre, il ne peut plus faire de nouveaux essais, mais il peut démarrer un nouveau jeu avec un nouveau nombre secret.

Pour avancés / 2GIN - Améliorations

Entrez la limite supérieure du jeu dans un champ texte. Le jeu est redémarré, lorsqu'on change la limite.

La limite ne peut pas être modifiée aussi longtemps que l'on n'a pas encore trouvé le nombre secret actuel.

Veillez à ce que la logique du jeu soit effectuée dans la classe modèle **`SecretNumber`** et non dans la vue (**`MainFrame`**). Pour cela, il est utile de modifier légèrement la classe **`SecretNumber`**. Dans **`MainFrame`**, réalisez une méthode **`reActivateComponents()`** pour activer et désactiver les composant selon l'état actuel du jeu.

Notions requises: MVC, new, typecast, String
Composants requis: JFrame, JTextField, JLabel
Structures requises: if

Exercice B.7: Les chaînes de caractères

Répondez aux questions suivantes en écrivant le code source respectif :

1. Déclaration d'une variable du type chaîne de caractères dénommée **`firstName`**.

   .....

2. Déclaration d'un attribut du type chaîne de caractères dénommé **`town`**, initialisé avec la valeur « Luxembourg ».

   .....

3. La variable **`weightValue`** (type **`String`**) contient la valeur « **`78`** ». Transformez cette valeur en un nombre entier et sauvegardez-la dans la variable **`weight`** (type **`int`**).

   .....

4. La variable **`age`** (du type **`double`**) contient la valeur 16.6. Transformez cette valeur en un **`String`** et sauvegardez-la dans la variable **`ageValue`** (type **`String`**).

   .....

5. Notez l'expression pour tester si le contenu de la variable **`blabla`** (type **`String`**) est égal à « **`vacances`** ».

   .....

6. Déterminez si le texte « **`pol`** » se trouve quelque part (c.-à-d. comme sous-chaîne) dans la chaîne de caractères **`text`** (du type **`String`**).

   .....

7. Déterminez si la variable **`comment`** (type **`String`**) contient comme sous-chaîne le texte qui se trouve actuellement dans la variable **`badWord`** (type **`String`**).

   .....

8. Déterminez si le texte dans la variable **`s1`** (type **`String`**) vient lexicographiquement avant le texte dans la variable **`s2`** (type **`String`**).

   .....

9. Affichez le contenu des variables **`wordA`** et **`wordB`** (type **`String`**) dans l'ordre lexicographique.

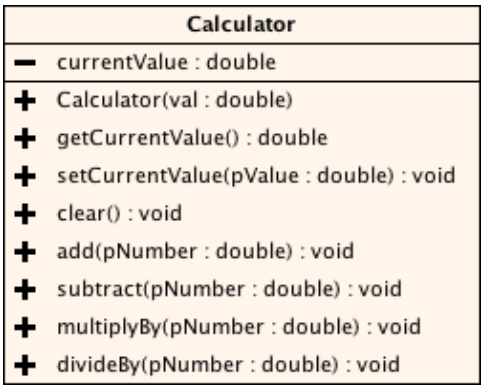
   .....

   .....

   .....

Classes requises: String, Integer, Double

Exercice B.8: Calculatrice



Reprenez l'exercice **`Calculator`** du cours de 3GIG.

Ajoutez à la classe **`Calculator`** le modificateur **`setCurrentValue`** qui permet d'assigner une valeur au champ **`currentValue`**. Ajoutez aussi un constructeur pour initialiser la valeur ainsi qu'une méthode **`clear`** qui efface la valeur actuelle.

Créez en NetBeans avec des boutons et un champ texte **`currentValueTextField`** (**`JTextField`**) un interface pour une petite calculatrice qui fonctionne en notation polonaise inverse (NPI) appelée en anglais : **RPN** - **R**everse **P**olish **N**otation.

Un certain nombre de calculatrices scientifiques et financières fonctionnent selon ce principe. Une caractéristique distinctive de ces calculatrices est la touche 'Enter' (au lieu d'une touche '=').

En notation polonaise inverse les opérandes précèdent les opérateurs.

Exemples : Calculs avec une calculatrice RPN	
Pour calculer 5 + 7, on fait :	**5 Enter 7 +**
Pour calculer 3 * (5+7) on fait :	**5 Enter 7 + 3 ***
On n'a donc pas besoin de parenthèses pour faire ces calculs sur une calculatrice RPN.	

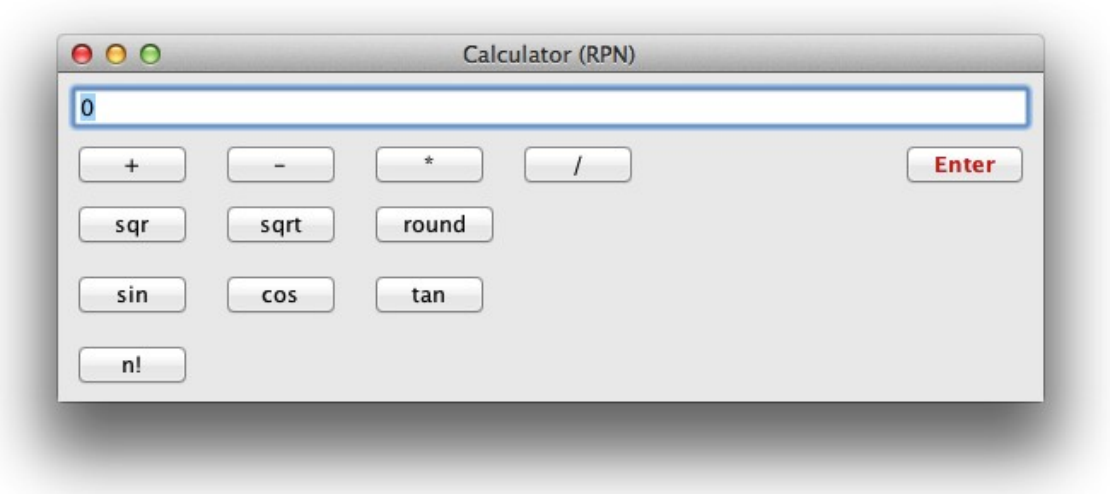
Interface :

Nommez les boutons selon les conventions (**`addButton`**, **`multiplyButton`**, …).

Le champ **`currentValueTextField`** montre la valeur actuelle de **`Calculator`** après chaque opération et il est aussi utilisé pour entrer les valeurs à ajouter, soustraire etc. L'instance de **`Calculator`** s'appellera **`calc`**.

Ajoutez ensuite dans la classe **`Calculator`** la possibilité d'effectuer les opérations suivantes sur **`currentValue`** : arrondir, calculer le carré, la racine carrée, le sinus, le cosinus, la tangente (employez les fonctions de la classe **`Math`**). Ajoutez ensuite des boutons à votre interface pour exécuter les nouvelles méthodes de **`Calculator`**.

Ajoutez à **`Calculator`** la possibilité de calculer la factorielle **n!** (→ voir 11e) de **`currentValue`**. Ajoutez un bouton pour intégrer le calcul dans votre interface. Arrondissez vers le bas et prenez la valeur absolue avant d'effectuer le calcul. Est-ce qu'il y a une limite pour le calcul de la factorielle ? Si oui, laquelle ?



Pour avancés / 2GIN :

Dans une copie du projet, essayez de réaliser une calculatrice avec le fonctionnement qui vous est connu de votre calculatrice de poche, avec une touche '=' pour effectuer les calculs. Dans ce cas, la suite des opérations est différente. Vous rencontrerez un petit problème lors des opérations à deux données (addition, soustraction, multiplication, division). Lequel ? Essayez de trouver une solution !

Notions requises :	Création de nouveaux objets, importation de projets Unimozer dans Netbeans, création de projets dans Netbeans, la classe Math, conversions : String ↔ valeurs numériques
Composants requis:	JButton, JTextField

Exercice B.9: Fraction

Une fraction se compose d'un numérateur et d'un dénominateur. En vous basant sur l'exercice réalisé comme répétition du cours de 11e (classe **`Fraction`**), développez un programme à interface graphique permettant les opérations suivantes :

- simplification d'une fraction
- multiplication d'une fraction par une autre
- division d'une fraction par une autre
- addition d'une fraction à une autre
- soustraction d'une fraction d'une autre
- affichage de la valeur décimale

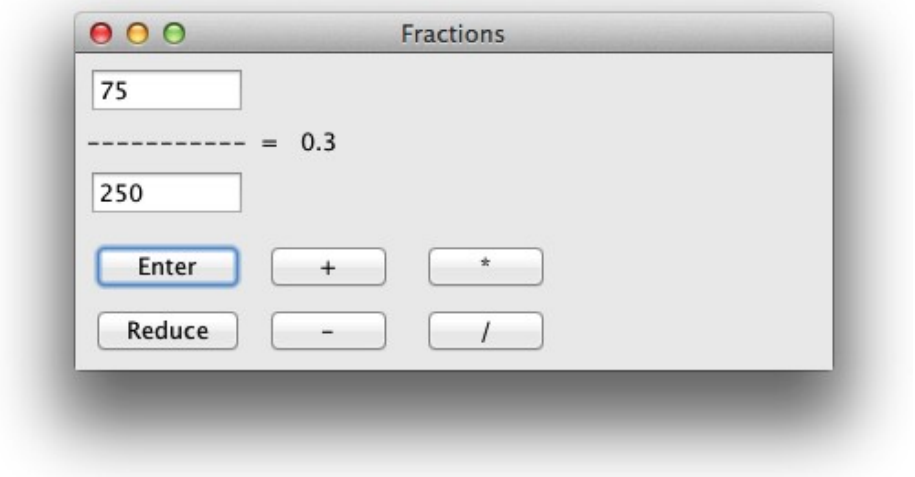
Employez un principe de fonctionnement similaire à celui de l'exercice **`Calculator`** (RPN). C.-à-d. Dans le programme il faut une instance **`fraction`** pour la fraction principale. La touche **Enter** sert à passer la première valeur à cette instance et le bouton **Reduce** permet de la simplifier.

Les boutons +, -, *, / permettent d'effectuer des calculs entre **deux** fractions.

Exemple : le bouton '+' ajoute la fraction qui a été entrée dans les champs d'édition à la fraction **`fraction`**. Le résultat (le contenu de la fraction **`fraction`**) est ensuite affiché dans les champs d'édition. Pour ce faire, il faut créer temporairement une deuxième instance du type **`Fraction`** qui est envoyée comme paramètre à la méthode **`add`** de **`fraction`**.

Améliorations pour simplifier le code :

- Développez une méthode **`updateView()`** qui transfère les données numériques du modèle (instance **`fraction`** de la classe **`Fraction`**) vers les 2 champs d'édition et le libellé.
- Développez une méthode **`getFractionFromView()`** qui retourne comme résultat une nouvelle fraction créée avec les données (numérateur et dénominateur) des champs d'édition.
- Appelez ces deux méthodes à tous les endroits appropriés de votre programme.



Notions requises: MVC, new, typecast
Composants requis: JFrame, JTextField, JButton

Exercice B.10: Le jeu du loup

Réalisez le jeu suivant selon le modèle MVC :

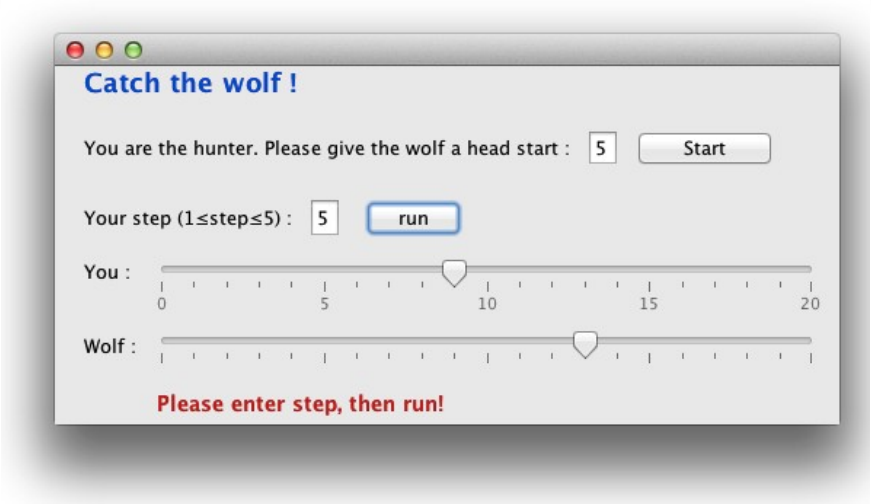
Un joueur essaie d'attraper un loup. Pour gagner, il doit se retrouver exactement à la position du loup. Avant de commencer la course, le joueur donne au loup une certaine avance. A chaque étape, le joueur indique le nombre de pas à faire pour avancer, tout en sachant que le loup avancera lui-même de 1 à 10 pas aléatoirement.

Le jeu se termine avec le message *You caught the wolf !* ou le message *The wolf escaped !* si l'un des coureurs à quitté la piste avant.

Réalisez d'abord comme modèle (MVC) une classe **`RunningTrack`** qui fait la gestion et l'évaluation des positions des deux adversaires et de leurs mouvements. Vous pouvez développer et tester la classe dans Unimozer.

Développez ensuite un interface en NetBeans qui permet de contrôler et d'afficher le déroulement du jeu.

Idée : Employez deux composants du type **`Slider`** ou **`ProgressBar`** pour afficher les positions des adversaires.



Notions requises: MVC, new, random
Composants requis: JFrame, JTextField, JButton, JLabel, JSlider