


## Introduction

Nous avons lors des séances précédentes conçu l'interface graphique de notre convertisseur d'unités. L'interface graphique développée en FXML et le contrôleur en [JavaFX](#) représentent respectivement la Vue (V) et le Contrôleur (C) dans une architecture MVC. Il nous reste donc à implémenter le **Modèle (M)** qui interagira avec le Contrôleur.

## Convertisseur

Le convertisseur d'unités doit fonctionner comme suit :

- On choisit le type de grandeur : Par exemple *Longueur, vitesse, température, etc.*
- On choisit l'unité source : Par exemple *mètres, yard, mille, année lumière, etc.*
- On choisit l'unité destination.
- Puis l'on tape une valeur dans l'unité source et on obtient la valeur dans l'unité destination.
- Le bouton  permet d'échanger les unités sources et destination lorsque cela est possible sans toutefois échanger les valeurs. Si cet échange n'est pas possible (on verra que c'est le cas avec certaines unités), ce bouton doit être désactivé.

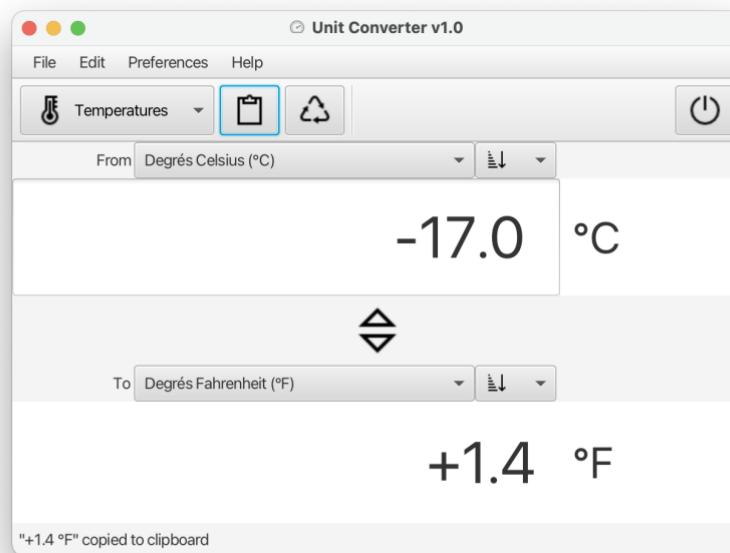


Figure 1 : Exemple de conversion de températures

Notre modèle de données sera implémenté dans la classe `Converter` et doit donc contenir :

- Le *type de grandeur des unités*. Ce qui permettra de construire une liste d'unités source et une liste d'unités destinations.
- Les *unités sources* avec parmi elles celle qui est actuellement sélectionnée comme *unité source*.
- Les *unités destinations* avec parmi elles celle qui est actuellement sélectionnée comme *unité destination*.

- Puisque l'on veut pouvoir trier les listes d'unités (suivant un ordre défini dans l'enum `SortOrder`), notre modèle devra contenir l'*ordre courant pour les unités sources* et l'*ordre courant pour les unités destinations*.
- Notre modèle doit aussi manipuler du texte :
  - En entrée pour “parser” une valeur pour l'unité source. Une fois parsée une valeur a été mise en place dans l'unité source. On peut alors la “formater” pour pouvoir afficher cette valeur.
  - En sortie, lorsqu'une valeur a été mise en place dans l'unité source, on peut alors déclencher la conversion de cette valeur dans l'unité destination et là aussi formater cette valeur.
  - Notre modèle pourra donc gérer deux chaînes : *la chaîne d'entrée* et *la chaîne de sortie*.
- Et enfin, on doit savoir si les unités source et destination sont interchangeables, ou plutôt si elles ne sont **pas** interchangeables : *unités non interchangeables*.

## Propriétés du Converter

Comme notre modèle de données (la classe `Converter`) va interagir avec le Contrôleur de la classe `Controller` qui utilise [JavaFX](#) nous allons simplifier les interactions entre les deux classes en utilisant des Propriétés ([JavaFX Property](#)). Les propriétés de notre modèle pourront alors être liées (unidirectionnellement ou bidirectionnellement) aux propriétés des éléments de l'UI.

- On peut obtenir la valeur d'une propriété avec la méthode : `E get()`
- On peut mettre en place une nouvelle valeur dans une propriété avec la méthode : `void set(E value)`
- On peut lier une propriété `p1` unidirectionnellement à une propriété `p2` avec la méthode `void bind(ObservableValue<? extends T> observable)` afin que les changements dans la propriété `p2` soient automatiquement propagés dans la propriété `p1`: `p1.bind(p2)`.
  - Attention toutefois, si une propriété est ainsi liée unidirectionnellement à une autre il est alors illégal de tenter de mettre en place une nouvelle valeur dans cette propriété liée. On peut toutefois savoir si une propriété est liée unidirectionnellement à une autre avec la méthode boolean `isBound()`
- On peut lier une propriété `p1` bidirectionnellement à une propriété `p2` avec la méthode `void bindBidirectional(Property<T> other)` afin que les changements dans d'une ou l'autre des propriétés soient automatiquement propagés dans l'autre ou l'une : `p1.bindBidirectional(p2)`

Si l'on reprend la liste des attributs du `Converter` énoncée plus haut on peut donc la revisiter en exprimant chaque élément sous la forme d'une propriété et comment cette propriété peut être liée à une propriété de l'UI :

- *Le type de grandeur des unités* : `ObjectProperty<MeasureType> measureType` qui peut être lié à la propriété `valueProperty` du `ComboBox<MeasureType> measuresComboBox` afin que les changements de grandeur dans le couple `Controller / Vue` puissent être propagés dans la propriété `measureType` du `Converter`.
- La liste des *unités sources* : `ObservableList<Unit<Double>> sourceUnits` qui peut être utilisé pour définir le contenu du `ComboBox<Unit<Double>> sourceUnitComboBox` dans le `Controller`.
- *L'unité source* : `ObjectProperty<Unit<Double>> sourceUnit` qui peut être lié à la propriété `valueProperty` du `ComboBox<Unit<Double>> sourceUnitComboBox`.
- La liste des *unités destinations* : `ObservableList<Unit<Double>> destinationUnits` qui peut être utilisé pour définir le contenu du `ComboBox<Unit<Double>> destinationUnitComboBox` dans le `Controller`.
- *L'unité destination* : `ObjectProperty<Unit<Double>> destinationUnit` qui peut être lié à la propriété `valueProperty` du `ComboBox<Unit<Double>> destinationUnitComboBox` dans le `Controller`.
- *L'ordre courant pour les unités sources* : `ObjectProperty<SortOrder> sourceSortOrder`
- *L'ordre courant pour les unités destinations* : `ObjectProperty<SortOrder> destinationSortOrder`
- *la chaîne d'entrée* : `StringProperty inputText` qui peut être lié bidirectionnellement à la `textProperty` du `TextField sourceTextField` du `Controller`.

- *la chaîne de sortie* : `StringProperty` `outputText`. La propriété `textProperty` du `Label` `destinationLabel` du `Controller` pourra être liée à cette propriété du `Converter` afin que celui-ci puisse formater la valeur dans l'unité destination lors de la conversion.
- *unités non interchangeables* : `BooleanProperty` `unexchangeableUnits`. La propriété `disableProperty` du `Button` `switchButton` du `Controller` pourra être liée à cette propriété du `Converter` afin que celui-ci puisse activer ou désactiver ce bouton en fonction de l'unité destination choisie.

L'endroit privilégié pour effectuer ces liaisons entre le `Converter` et le `Controller` sera dans la méthode `initialize` du `Controller`.

Les différentes méthodes pour relatives à une propriété du `Converter`, par exemple pour la propriété `ObjectProperty<...> xxx` sont les suivantes:

- `ObjectProperty<...> xxxProperty()` : pour obtenir la propriété (afin de la lier par exemple avec `bind` ou `bindBidirectional`).
- `... getXxx()` pour obtenir la valeur de la propriété avec `xxx.get()`.
- `void setXXX(... value)` pour mettre une nouvelle valeur dans la propriété `xxx`. Notez que si cette propriété a été liée unidirectionnellement (avec la méthode `bind`), il sera alors interdit de faire un `set(value)` de cette propriété.
- `void applyXxx()` applique toutes les actions à effectuer une fois que la propriété `xxx` a changé de valeur :
  - soit au travers de la méthode `setXxx(... value)`
  - soit parce que la propriété `xxx` a été liée à une propriété extérieure.

## Types d'unités

On peut distinguer les unités de base (typiquement les [unités SI](#)) et les autres unités dans lesquelles on veut pouvoir convertir les valeurs des unités SI.

Toutes les unités que nous allons utiliser possèdent une valeur qui la plupart du temps est un nombre. Les unités doivent pouvoir créer une chaîne de caractère pour soit afficher ce nombre (avec un certain formatage), néanmoins certaines unités (dites unités **symboliques**) affichent un symbole plutôt qu'un nombre. C'est le cas notamment de l'[échelle de Beaufort](#) qui mesure la vitesse du vent (e.g. 45 km/h  $\Leftrightarrow$  "Vent frais") ou bien des directions cardinales (Nord, Nord-Est, etc.)

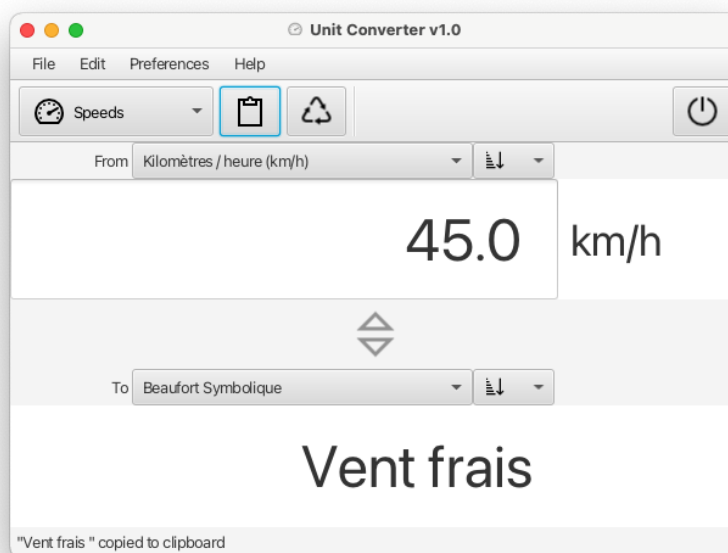
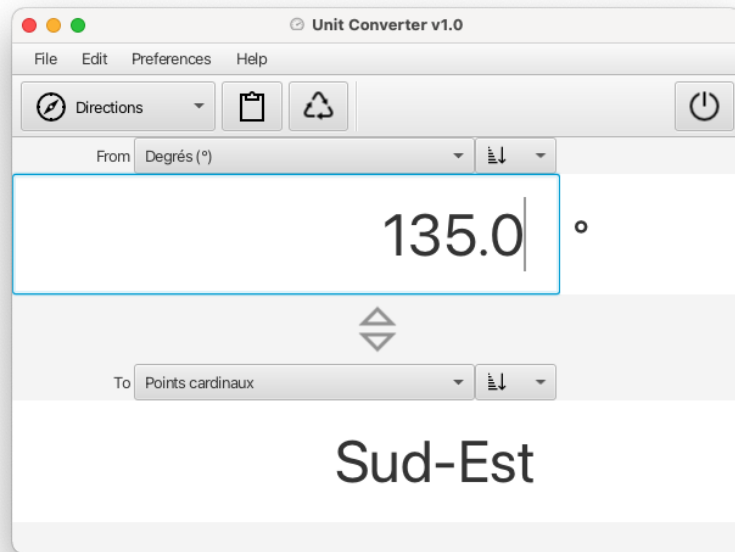
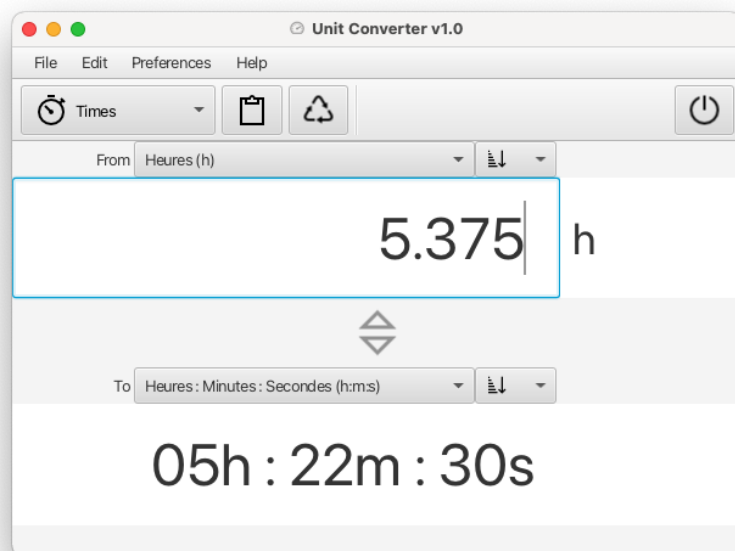


Figure 2 : Unité Symbolique Beaufort



*Figure 3 : Unité Symbolique Directions cardinales*

Certaines unités doivent permettre de décomposer une valeur en plusieurs valeurs. C'est le cas notamment lorsque l'on souhaite décomposer une valeur de temps exprimée en heures en heures : minutes : secondes, ou bien encore une valeur d'angle exprimée en degrés en degrés, minutes, secondes.



*Figure 4 : Unité Décomposée*

On pourra noter que les unités symboliques comme les unités décomposées ne sont **pas** interchangeables avec les unités numériques et ne feront donc pas partie des unités sources car on ne peut pas simplement saisir un nombre pour mettre en place une valeur dans ces unités.

Le diagramme de classes ci-dessous résume l'implémentation proposée des différentes unités :

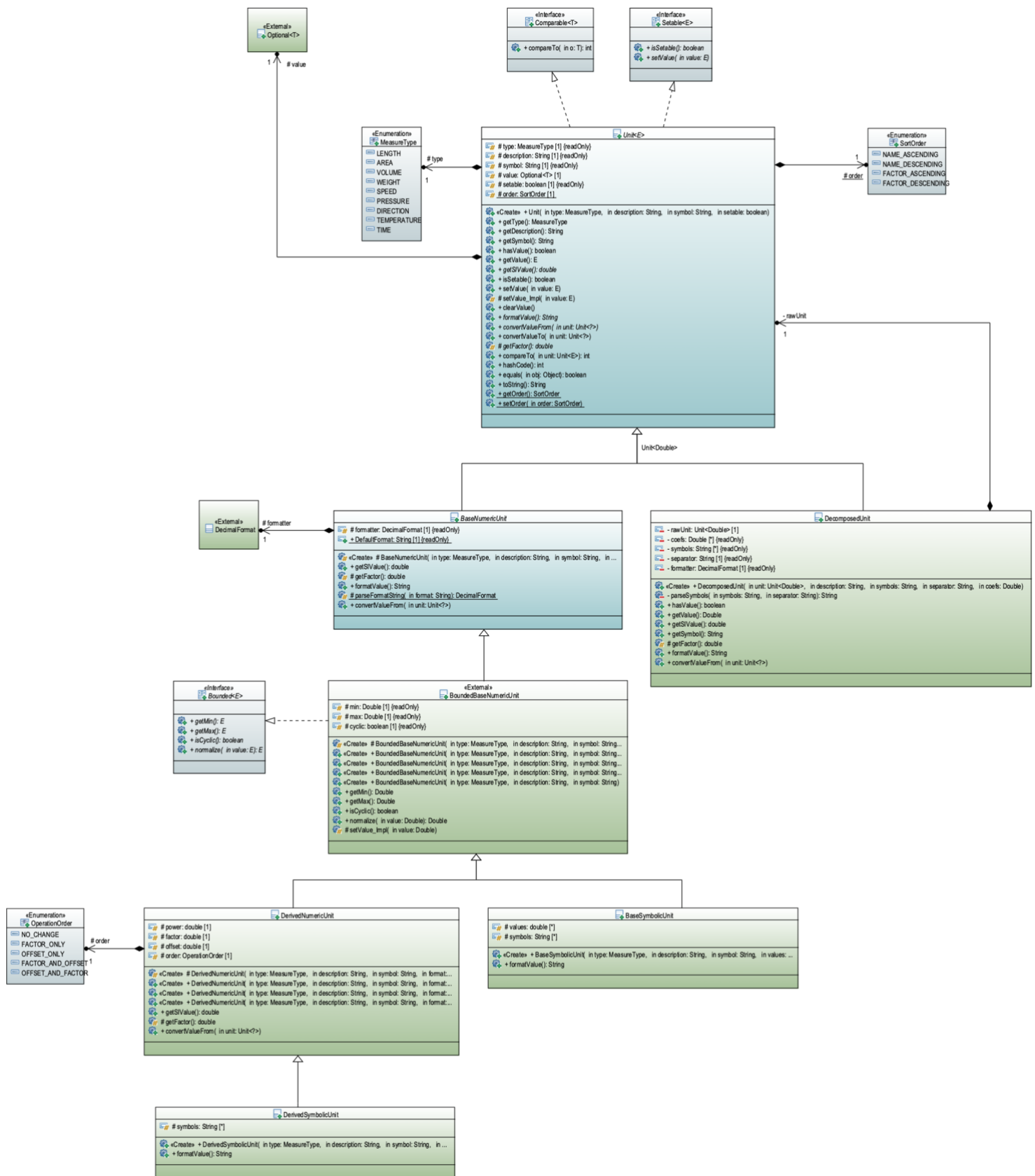


Figure 5 : Hiérarchie des classes Unités

- Unit<E> : classe mère abstraite qui implémente 2 interfaces :
  - Comparable<Units<?>> afin que l'on puisse comparer les unités entre elles au sens du 3way compare (-1, 0, +1) (pour les trier notamment).
  - Setable<E> qui permet de savoir si on peut (ou pas, avec la méthode isSetable()) utiliser la méthode setValue(E value) pour mettre en place une nouvelle valeur dans l'unité. Typiquement les unités symboliques ou décomposées (voir plus haut) peuvent afficher une valeur, mais ne permettent pas de "saisir" une nouvelle valeur.

- **BaseNumericUnit** implémentation partielle de `Unit<Double>` pour les unités de base (type unité SI) qui contient de quoi :
  - Fournir une valeur de SI : c'est ce qui nous servira dans toutes les unités pour les conversions.
  - Convertir une valeur d'une autre unité en lui demandant sa valeur SI.
  - Formater une valeur Double avec un `DecimalFormat`.
- **BoundedBaseNumericUnit** : première implémentation concrète pour les unités de base implémentant l'interface `Bounded<E>` permettant de définir des valeurs min et max pour la valeur de l'unité. Ce range [min...max] peut éventuellement être cyclique comme les angles entre 0 et 360°. D'autre part, si une unité n'a pas de borne supérieure pour ses valeurs on pourra alors utiliser la valeur `Double.POSITIVE_INFINITY`.
- **BaseSymbolicUnit** : version spéciale de `BoundedBaseNumericUnit` qui au lieu de formater un nombre va utiliser des tableaux de symboles et de valeurs : la valeur de l'unité déterminera grâce au tableau de valeurs quel symbole du tableau de symboles devra être affiché dans la méthode `formatValue()`. Peut être utilisé pour les unités symboliques telles que l'[échelle de Beaufort](#) ou bien les directions cardinales (Nord, Nord-Est, etc.)
- **DerivedNumericUnit** : Implémentation concrète pour les unités dérivées dont la valeur peut être calculée par rapport à une unité de base (avec une puissance, un facteur et un offset).
  - L'enum `OperationOrder` définit comment utiliser la puissance, le facteur et l'offset ainsi que les opérations à réaliser.
  - la puissance, le facteur et l'offset définissent comment convertir une valeur exprimée dans l'unité SI de cette grandeur vers la valeur d'une unité dérivée. Voir [Grandeurs et Unités](#) en fin de document.
- **DerivedSymbolicUnit** : version spéciale de `DerivedNumericUnit` qui au lieu de formater un nombre va utiliser un tableau de symboles : la valeur déterminera quel symbole devra être affiché dans la méthode `formatValue()`. Peut aussi être utilisé pour les unités symboliques. Cette classe joue le même rôle que la classe `BaseSymbolicUnit` mais possède une valeur interne puisqu'elle hérite de `DerivedNumericUnit`.
- **DecomposedUnit** une unité spéciale permettant de décomposer une unité (contenant une seule valeur) en plusieurs valeurs permettant ainsi de transformer par exemple une durée en heures (e.g. 5.475 h) en heure : minutes : secondes (05h : 28m : 29s). Il n'y a en fait pas de décomposition, juste un formatage particulier dans `formatValue()`.
- **UnitsFactory** est une classe fabrique permettant de générer toutes les unités de différentes grandeurs avec sa méthode `Set<Unit<Double>> getUnits(MeasureType type)` que vous devrez compléter.

## Travail à réaliser

1. Importez le projet "Units Model" dans votre IDE préféré : l'archive se trouve dans `/pub/FISE_LAOB12/TP4/UnitsModel.zip`.
2. Complétez les différents TODO dans les diverses unités et dans la classe `Converter`.
3. Complétez la classe `UnitsFactory` avec les différentes unités listées dans [Grandeurs et Unités](#).
4. Vous pourrez vérifier le bon fonctionnement de vos unités et du `Converter` avec les programmes contenus dans les classes :
  - `TestUnits`
    - Nécessite `-ea` (enable assertions) comme argument de VM.
  - `TestConverter`
    - La classe `TestConverter` utilisant les propriétés de JavaFX, elle nécessite aussi les arguments `-ea --module-path <votre path vers JavaFX> --add-modules javafx.base` dans les arguments de VM.
5. Intégrez votre modèle finalisé à votre projet d'interface graphique réalisé lors de la dernière séance. Simplement en copiant/collant vos classes d'un projet à l'autre (en respectant les packages sources et destination).



6. Complétez votre classe Controller :

1. Ajoutez une instance de Converter parmi les attributs non FXML du Controller
2. Créez les liaisons entre le Converter et le Controller dans la méthode initialize.
3. Complétez les différents callbacks du Controller (les méthodes on...Action(ActionEvent event)) en utilisant autant que faire se peut les méthodes de votre instance de Converter.
4. Ajoutez un nouveau callback pour afficher une simple boîte de dialogue présentant l'application et que vous relierez à l'item de menu "About..." de l'interface graphique :

```
public void onAboutAction(ActionEvent event)
{
    logger.info("About action triggered");

    ...

    Alert alert = new Alert(...);

    ...

    alert.showAndWait();
}
```

Vous aurez jusqu'au mardi 09/05/2023 23:59 pour déposer votre projet sur [exam.ensiie.fr](https://exam.ensiie.fr) dans le dépôt **laob12-converter**.

## Grandeurs et Unités

Les différentes unités de notre convertisseur appartiennent à plusieurs grandeurs (représentées par l'enum Measures dans notre projet) : Les longueurs ou distances, les aires, les volumes, les masses, les vitesses, les pressions, les directions, les températures et les durées.

## Notations

$valetur = [partie_{entiere}] + \{partie_{decimale}\}$

## Longueurs / distances

- Mètre ( $m$ ) :  $l_m$ 
  - $min = 0$
  - $max = \infty$
- Centimètre ( $cm$ ) :  $1\ cm = \frac{1}{100}\ m = 0.01\ m$
- Millimètre ( $mm$ ) :  $1\ mm = \frac{1}{1000}\ m = 0.001\ m$
- Pouce ( $inch$  ou  $"$ ) :  $1\ in = 0.0254\ m$
- Pied ( $ft$  ou  $'$ ) :  $1\ ft = 0.3048\ m$
- Yard ( $yd$ ) :  $1\ yd = 0.9144\ m$
- Kilomètre ( $km$ ) :  $1\ km = 1000\ m$
- Mille ( $mi$ ) :  $1\ mi = 1609.344\ m$
- Mille Nautique ( $nM$ ) :  $1\ nM = 1852\ m$

## Surfaces

- Mètre carré ( $m^2$ )
  - $min = 0$
  - $max = \infty$
- Centimètre carré ( $cm^2$ ) :  $1\ cm^2 = \frac{1}{100 \times 100}\ m^2 = 0.0001\ m^2$
- Hectare ( $ha$ ) :  $1\ ha = 1\ hm^2 = 100 \times 100\ m^2 = 10000\ m^2$
- Square Inch ( $in^2$ ) :  $1\ in^2 = 6.4516\ cm^2 = 0.00064516\ m^2$

- Square foot ( $ft^2$ ) :  $1 ft^2 = 0.09290304 m^2$
- Square yard ( $yd^2$ ) :  $1 yd^2 = 0.83612736 m^2$
- Acre (ac) :  $1 ac = 4046.8564224 m^2$
- Kilomètre carré ( $km^2$ ) :  $1 km^2 = 1000 \times 1000 m^2 = 1000000 m^2$

## Volumes

- Mètre cube ( $m^3$ ) :  $v_{m^3}$ 
  - $min = 0$
  - $max = \infty$
- Litre ( $l$ ) :  $1 l = \frac{1}{1000} m^3 = 0.001 m^3$
- Gallon ( $gal$ ) :  $1 gal = 4.54609 l = 0.00454609 m^3$
- US Gallon ( $USgal$ ) :  $1 USgal = 3.785411784 l = 0.003785411784 m^3$
- Centilitre ( $cl$ ) :  $1 cl = \frac{1}{100} l = \frac{1}{100000} m^3 = 0.00001 m^3$
- Millilitre ( $ml$ ) :  $1 ml = \frac{1}{1000} l = \frac{1}{1000000} m^3 = 0.000001 m^3$
- Centimètre cube ( $cm^3$ ) :  $1 cm^3 = 1 ml = \frac{1}{1000000} m^3 = 0.000001 m^3$
- Ounce (oz) :  $1 oz = 29.5735295625 ml = 0.00002957352956 m^3$
- Cubic foot ( $ft^3$ ) :  $1 ft^3 = 0.02831685 m^3$
- Cuillère à café / tea spoon ( $tsp$ ) :  $1 tsp \approx 5ml = 0.000005 m^3$
- Cuillère à soupe / table spoon ( $tbsp$ ) :  $1 tbsp \approx 15ml = 0.000015 m^3$

## Masses

- Kilogramme ( $kg$ ) :  $m_{kg}$ 
  - $min = 0$
  - $max = \infty$
- Gramme ( $g$ ) :  $1 g = \frac{1}{1000} kg$
- Milligramme ( $mg$ ) :  $1 mg = \frac{1}{1000000} kg$
- Livre / Pound ( $lb$ ) :  $1 lb = 0.45359237 kg$
- Tonne (t) ( $t$ ) :  $1 t = 1000 kg$
- Long ton (LT) :  $1 LT = 1016.047 kg$

## Vitesses

- Mètre par seconde ( $m/s$ ) :  $v_{m/s}$ 
  - $min = 0$
  - $max = \infty$
- Kilomètre par heure ( $km/h$ ) :  $1 km/h = \frac{1}{3.6} m/s$
- Mile par heure ( $mph$ ) :  $1 mph = 0.44704 m/s$
- Mile per minute ( $mpm$ ) :  $1 M/min = \frac{1}{60} mph = 0.007450666667 m/s$
- Nœud ( $kn$ ) :  $1 kn = 0.514444 m/s$
- Minutes par Kilomètre ( $min/km$ ) :  $v_{min/km} = \frac{100}{v_{m/s} \times 6} = \frac{16.6666}{v_{m/s}}$
- Minutes : Secondes par Kilomètre ( $m : s/km$ ) :  $[v_{min/km}] : [\{v_{min/km}\} \times 60]$
- Minutes par Mille ( $min/mi$ ) :  $v_{min/mi} = \frac{223.693629}{v_{m/s} \times 6} = \frac{37.2822715}{v_{m/s}}$
- Échelle de Beaufort (?) :  $v_{Beaufort} = \lfloor \sqrt[3]{\frac{v_{km/h}^2}{9}} \rfloor = \lfloor \sqrt[3]{1.44 \times v_{m/s}^2} \rfloor \Rightarrow v_{m/s} \approx 0.83 \times v_{Beaufort}^{\frac{3}{2}}$

Que l'on peut aussi représenter par des "symboles" comme décrits dans le tableau ci-dessous



	Symbole	$V_{min}(m/s)$	$V_{max}(m/s)$	Effets
0	Calme	0	0.28	la fumée s'élève verticalement ; la mer est comme un miroir
1	Très légère brise	0.28	1.53	il se forme des rides, mais il n'y a pas d'écume
2	Légère brise	1.53	3.19	vaguelettes courtes ; leurs crêtes ne déferlent pas
3	Petite brise	3.19	5.42	très petites vagues ; écume d'aspect vitreux
4	Jolie brise	5.42	7.92	petites vagues devenant plus longues ; moutons nombreux
5	Bonne brise	7.92	10.69	vagues modérées, allongées ; moutons nombreux
6	Vent frais	10.69	13.75	des lames se forment ; crêtes d'écume blanche plus étendues
7	Grand frais	13.75	17.08	la mer grossit ; l'écume est soufflée en traînées ; lames déferlantes
8	Coup de vent	17.08	20.69	lames de hauteur moyenne ; de leurs crêtes se détachent des tourbillons d'embruns
9	Fort coup de vent	20.69	24.31	grosses lames ; leur crête s'écroule et déferle en rouleaux
10	Tempête	24.31	28.47	grosses lames à longues crêtes en panache ; déferlement en rouleaux intense et brutal
11	Violente tempête	28.47	32.64	lames exceptionnellement hautes ; mer recouverte de bancs d'écume blanche
12	Ouragan	32.64	$\infty$	air plein d'écume et d'embruns ; mer entièrement blanche ; visibilité très réduite

## Pressions

- **Pascals** ( $Pa = N/m^2$ ) :
  - $min = 0$
  - $max = \infty$
- hectopascals ( $hPa$ ) :  $1 hPa = 100 Pa$
- **Atmosphère** ( $atm$ ) :  $1 atm = 101325 Pa$
- **Bar** ( $bar$ ) :  $1 bar = 100000 Pa$
- Millibars ( $mbar$ ) :  $1 mbar = 100 Pa$
- **Millimètres de mercure** ( $mm Hg$ ) :  $1 mm Hg = 133.3224 Pa$
- **Pouces de mercure** ( $inch Hg$ ) :  $1 inch Hg = 3386.39 Pa$
- **Pounds per square inch** ( $psi$ ) :  $1 psi = 6894.757 Pa$

## Directions

- Radians ( $\pi$ ):  $d_r$ 
  - $min = 0$
  - $max = 2\pi$
- Degrés ( $^\circ$ ) :  $1^\circ = \frac{\pi}{180} \text{ radians}$ 
  - $min = 0$
  - $max = 360$
- Degrés, minutes, secondes ( $^\circ, ', ''$ ) :  $d_d = \lfloor d_d \rfloor + \{d_d\}$ 
  - $min = 0^\circ, 0', 0''$
  - $max = 359^\circ, 59', 59''$
  - $d^\circ = \lfloor angle^\circ \rfloor$
  - $d' = \lfloor \{angle^\circ\} * 60 \rfloor$
  - $d'' = \lfloor \{\{angle^\circ\} * 60\} * 60 \rfloor$
- Directions symboliques :
  - Nord =  $0 \pm \frac{\pi}{8}$
  - Nord-Est =  $\frac{\pi}{4} \pm \frac{\pi}{8}$
  - Est =  $\frac{\pi}{2} \pm \frac{\pi}{8}$
  - ...,
  - Nord-Ouest =  $\frac{7\pi}{4} \pm \frac{\pi}{8}$

## Températures

- Degrés Kelvin ( $^\circ K$ ) :  $t_K$ 
  - $min = 0^\circ K$
  - $max = \infty^\circ K$
- Degrés Celsius ( $^\circ C$ ) :  $t_C = t_K + 273.15$ 
  - $min = -273.15^\circ C$
  - $max = \infty^\circ C$
- Degrés Fahrenheit ( $^\circ F$ ) :  $t_F = (t_K + 459.67) \times \frac{5}{9}$

## Temps

- seconde (s) :  $t_s$ 
  - $min = 0$
  - $max = \infty$
- minute (min) :  $t_m : 1 \text{ min} = 60 \text{ s}$
- heure (h) :  $t_h : 1 \text{ h} = 3600 \text{ s}$
- minute : secondes (m:s) à partir de  $t_s$  :  $t_{m:s} = \lfloor t_s \times \frac{1}{60} \rfloor : \lfloor \{t_s \times \frac{1}{60}\} \times 60 \rfloor$
- minute : secondes (m:s) à partir de  $t_m$  :  $t_{m:s} = \lfloor t_m \rfloor : \lfloor \{t_m\} * 60 \rfloor$
- heure : minute : seconde (h:m:s) à partir de  $t_s$  :
  - heures :  $\lfloor t_s \times \frac{1}{3600} \rfloor$
  - minutes restantes :  $\lfloor \{t_s \times \frac{1}{3600}\} \times 60 \rfloor$
  - secondes restantes :  $\lfloor \{\{t_s \times \frac{1}{3600}\} \times 60\} \times 60 \rfloor$
- heure : minute : seconde (h:m:s) à partir de  $t_h$  :
  - heures :  $\lfloor t_h \rfloor$
  - minutes restantes :  $\lfloor \{t_h\} * 60 \rfloor$
  - secondes restantes :  $\lfloor \{\{t_h\} * 60\} * 60 \rfloor$