

MVC (Model – View – Controller)

En général des programmes avec une interface graphique possèdent une structure MVC qui est expliquée par la suite de ce document.

1 Version standard

	Model (Modèle)	View (Vue)	Controller (Contrôleur)
Responsabilité	Données	Présentation des données	Synchronisation de la vue et du modèle
Analogie	Chaîne	Téléviseur	Télécommande

2 Version simplifiée (pour programmes très simples)

	Model (Modèle)	View (Vue) & Controller (Contrôleur)
Responsabilité	Données	Présentation des données Synchronisation de la vue et du modèle
Analogie	Chaîne	Téléviseur & Éléments de commande intégrés
Environnement de développement intégré	Unimozzer	NetBeans

3 Avantages du MVC

Le fait que le téléviseur, la télécommande et les chaînes sont complètement séparés permet de facilement remplacer la télécommande ou le téléviseur sans que les chaînes ne soient affectées en quelque sorte. En plus, quand une nouvelle chaîne est disponible, il ne faut pas remplacer le téléviseur ou la télécommande. On peut facilement remplacer et modifier un des composants à condition que l'interface présentée aux autres composants reste la même.

Des programmes MVC présentent des avantages similaires. Le fait que le modèle, le contrôleur et la vue sont complètement séparés permet de facilement remplacer un des composants sans affecter les deux autres composants, à condition que l'interface publique reste la même.

4 Grandes étapes pour créer un programme avec une structure MVC

1. Création du modèle dans Unimozzer
2. **Ouvrir (File -> Open Project)** le projet Unimozzer dans Netbeans (on peut simplement l'ouvrir comme les formats des projets Unimozzer et Netbeans sont les mêmes)
3. Ajout d'un **JFrame Form** (click droit sur **<default package>** -> **New -> JFrame Form**) nommé **MainFrame**.
4. Création de l'interface graphique par glisser – déposer (drag and drop). Adapter les étiquettes (labels) montrées à l'utilisateur.
5. Changer le nom des variables des éléments auxquels il faut accéder dans le code source
6. Ajout d'un attribut privé à la classe `MainFrame` pour lui permettre de communiquer avec le modèle. Ne pas oublier de l'initialiser en utilisant le constructeur.
7. Ecrire les méthodes qui définissent le comportement des boutons et des autres éléments graphiques.

Les parties 6 et 7 sont détaillées par la suite.

Attention ! Dans toutes les explications qui suivent les parties du code entre < et > doivent être remplacées par les informations appropriées dans vos exemples.

5 Communication Vue - Modèle

Cette partie explique comment déclarer et initialiser l'attribut qui permet à la vue de communiquer avec le modèle.

5.1 Syntaxe de la déclaration

```
private <nom de la classe> <nom de l'attribut>;
```

Le nom de l'attribut est au choix du programmeur.

Question à se poser : Quel est le nom de la classe modèle ?



5.2 Que se passe-t-il lors de la déclaration de l'attribut ?

MVC

Un dispositif permettant à la vue de communiquer avec un objet à la classe modèle est mis en place.

Analogie - Voiture & Remorque

Un dispositif d'attelage permettant d'attacher une remorque est mis en place.



5.3 Syntaxe de l'initialisation

```
<nom de l'attribut> = new <nom de la classe>(<paramètres>);
```

Les valeurs des paramètres sont séparées par des virgules. Leur ordre est le même que l'ordre des paramètres dans la définition du constructeur.

5.4 Que se passe-t-il après l'initialisation de l'attribut à l'aide du constructeur ?

MVC

Un objet concret de la classe modèle est créé et affecté à l'attribut déclaré avant. La vue peut maintenant accéder l'interface publique de l'objet modèle à travers le nom de l'attribut.

Analogie - Voiture & Remorque

Une remorque est attachée au dispositif d'attelage.



5.5 Comment tester si un attribut non primitif est initialisé ou non initialisé ?

Un attribut à type non primitif possède la valeur `null` tant qu'il n'a pas été initialisé. Pour tester si l'attribut a été initialisé il suffit donc de faire le test suivant :

```
if(<nom de l'attribut> == null){  
    //instructions si l'attribut n'est pas initialisé  
}else{  
    //instructions si l'attribut est initialisé  
}
```

Analogie Voiture & Remorque

Tant qu'une remorque n'est pas attachée au dispositif d'attelage, la valeur du dispositif est `null`.

5.6 Syntaxe d'une déclaration et initialisation combinées

```
private <nom de la classe> <nom de l'attribut>  
    = new <nom de la classe>(<paramètres>);
```

Parfois il est possible de déclarer et d'initialiser l'attribut en même temps. (voir section suivante)

5.7 Où faut-il initialiser l'attribut de la classe modèle ?

La question importante que nous devons nous poser est la suivante :

Est-ce qu'il nous faut des données supplémentaires de l'utilisateur pour pouvoir appeler le constructeur de la classe modèle ?



NON

On peut déclarer et initialiser l'attribut de la classe modèle en même temps.

OUI

Si oui, il faut déterminer comment l'utilisateur procède pour nous passer les données. En général, il va nous les entrer dans un ou plusieurs champs de texte puis cliquer sur un bouton. C'est alors dans la méthode qui nous permet de définir le comportement de ce bouton qu'il nous faut initialiser l'attribut de la classe modèle.

6 Définir le comportement des éléments graphiques (p. ex. boutons)

Le corps d'une méthode qui définit le comportement d'un bouton (ou d'un autre élément de contrôle graphique) peut être divisé en trois parties.

```
//partie générée automatiquement
{
    //Partie 1 : Entrées de l'utilisateur (Input)

    //Partie 2 : Traitements à exécuter (Processing)

    //Partie 3 : Affichage (Output)
}
```

6.1 Partie 1 : Entrées de l'utilisateur (Input)

La question importante à se poser ici est la suivante :

Est-ce qu'il y a des données fournies par l'utilisateur à travers des champs de texte ou à travers d'autres éléments graphiques qu'il faut extraire pour définir le comportement du bouton.

NON

Procéder au point 2.2.

OUI

S'il y a des données fournies par l'utilisateur, il faut d'abord extraire ces données et les stocker dans des variables.

6.1.1 Questions à se poser

- Quel est le type des données à extraire ?
- Quel est le nom du champ / de la glissière ?
- Quelle est la méthode pour accéder au contenu du champ / de la glissière ?

6.1.2 Extraire une donnée d'un champ de texte :

```
String <nom de variable> = <nom du champ>.getText();
int <nom de variable> = Integer.valueOf(<nom du champ>.getText());
double <nom de variable> = Double.valueOf(<nom du champ>.getText());
```

Attention !

Toute donnée qui se trouve dans un champ de texte est par défaut une chaîne de caractères (String). S'il s'agit d'une valeur entière (int) ou décimale (double), il faut la convertir à l'aide de la méthode `Integer.valueOf` respectivement `Double.valueOf`

6.1.3 Extraire la valeur d'une glissière (JSlider)

```
int <nom de variable> = <nom de la glissière>.getValue();
```

La valeur d'une glissière est toujours un nombre entier entre une valeur maximale et une valeur minimale.

6.2 Partie 2 : Traitements à exécuter (Processing)

Cette partie est impossible à généraliser. Elle varie d'un programme à l'autre. Souvent on fait des calculs dans cette partie et on adapte les données du modèle tout en utilisant les méthodes publiques du modèle.

Il y a cependant une question principale (à laquelle la réponse est souvent oui) qui permet de se guider.

Est-ce qu'il y a une ou plusieurs méthodes du modèle qui font ce que nous devons faire ?



NON

Il faut réaliser la fonctionnalité dans le contrôleur ou la vue.

OUI

Utiliser les méthodes du modèle. Pour appeler une telle méthode sans valeur retournée (void), la syntaxe est la suivante :

```
<nom de l'attribut modèle>.<nom de la méthode>(<paramètres>);
```

S'il s'agit d'une méthode qui retourne une valeur, on veut souvent stocker cette valeur dans une variable. L'instruction devient alors, selon le type de retour :

```
String <nom de variable> = <nom de l'attribut modèle>.<nom de la méthode>(<paramètres>);
```

```
int <nom de variable> = <nom de l'attribut modèle>.<nom de la méthode>(<paramètres>);
```

```
double <nom de variable> = <nom de l'attribut modèle>.<nom de la méthode>(<paramètres>);
```

Les paramètres sont souvent des valeurs extraites dans la partie **input**. Parfois ce sont aussi des constantes.

6.3 Partie 3 : Affichage des résultats (Output)

Dans cette partie-ci on affiche les résultats des traitements effectués.

6.3.1 Questions à se poser ?

- Quels sont les noms des labels, des glissières et des barres de progression dont les informations affichées doivent être actualisées ?
- Quelle est le nom de la méthode, souvent un manipulateur (setter), qu'il faut appeler pour actualiser l'information affichée ?
- Quelles sont les nouvelles informations à afficher ?



6.3.2 Affichage dans un label / une étiquette (JLabel)

```
<nom du label>.setText(<texte à afficher>);
```

Souvent le texte à afficher dépend des valeurs retournées par le modèle dans la partie **processing**. Il est très important de ne surtout pas utiliser les valeurs de la partie **input**, mais de passer par les accesseurs du modèle. Dans quelques cas il est suffisant d'utiliser les accesseurs du modèle, mais souvent il faut ajouter un texte plus expressif dans le contrôleur / la vue.

Il est important de noter que la méthode `setText` prend une chaîne de caractères en paramètre. Des valeurs entières ou décimales doivent être converties en chaîne de caractères avant d'être affichées. Ceci peut se faire à l'aide d'une des deux expressions suivantes :

1^{ère} expression :

```
" " + <valeur décimale ou entière>
```

2^{ème} expression :

```
String.valueOf(<valeur décimale ou entière>)
```

6.3.3 Affichage dans une barre de progression (JProgressBar)

```
<nom de la barre>.setValue(<valeur entière>);
```

Attention !

La valeur d'une barre de progression est **toujours** une **valeur entière** entre une valeur minimale et une valeur maximale.

6.3.4 Affichage dans une glissière (JSlider)

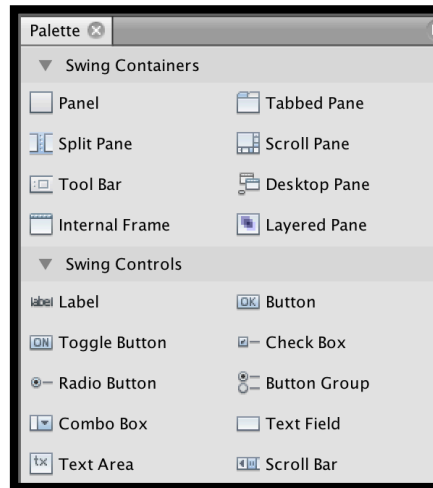
```
<nom de la glissière>.setValue(<valeur entière>);
```

Attention !

La valeur d'une glissière est **toujours** une **valeur entière** entre une valeur minimale et une valeur maximale.

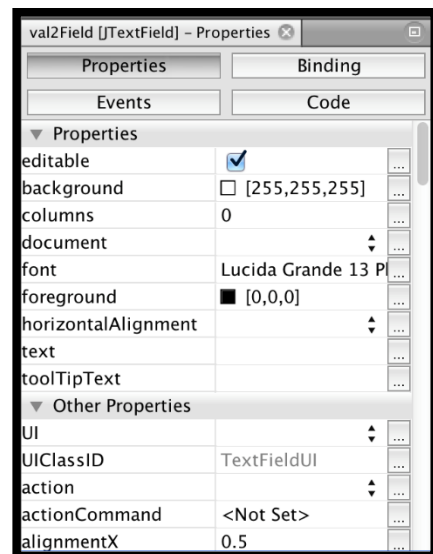
7 Kit de premier secours pour NetBeans

7.1 Comment afficher la palette avec les éléments graphiques ?



Cliquer sur les menus **Window > IDE Tools > Palette**

7.2 Comment afficher les propriétés des éléments graphiques ?



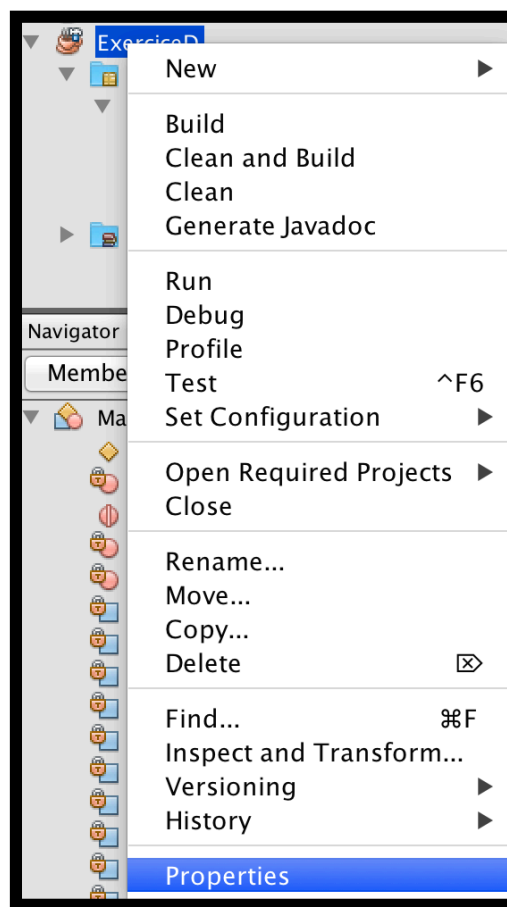
Cliquer sur les menus **Window > IDE Tools > Properties**

7.3 Comment afficher la barre de navigation "Source" / "Design" ?

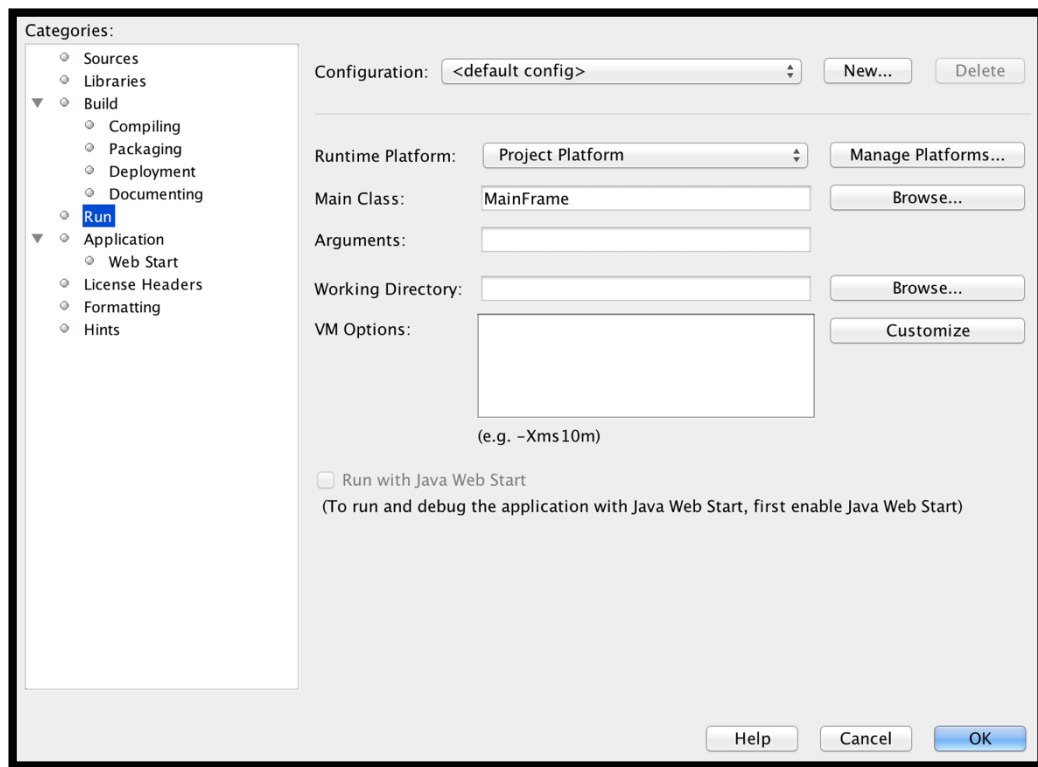


Cliquer sur les menus **View > Show Editor Toolbar**

7.4 Que faire si j'obtiens un message d'erreur qu'il n'y a pas de classe principale au démarrage de mon programme ?



1. Clic droit sur le nom du projet, puis choisir l'option **Properties** tout en bas du menu contextuel. La fenêtre suivante va s'ouvrir.



2. Cliquer sur **Run**, puis entrer le nom de la classe principale dans le champ de texte avec l'étiquette **Main Class** (mais **sans** l'extension **.java**)
3. Cliquer sur **OK**. Le programme devrait démarrer maintenant.