



Java

Version 0.7 (créé le 14/01/2024, modifié le 21/05/2024)

Java est un langage de programmation de haut niveau orienté objet qui reprend en grande partie la syntaxe du langage C++. Néanmoins, Java est épuré des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que les pointeurs et références, ou l'héritage multiple contourné par l'implémentation des interfaces.

Outils nécessaires :

- Un logiciel de codage (Visual Studio Code ou Notepad++)
- Java JDK
- Ou un logiciel tout en un (NetBeans (fortement recommandé))

Table des matières :

I. Bases

I.1. Syntaxe

I.2. Enregistrer et exécuter un programme `NomDuProgramme.java`

I.3. Types de variables

I.3.1. Numériques

I.3.2. Caractères

I.3.3. Autres types

I.4. Commentaires

I.5. Opérations

I.6. Les variables

I.7. Les tableaux

I.8. Conditions

I.8.1. Opérateurs de comparaison

I.8.2. Tests de conditions

I.9. Boucles

II. Les méthodes et les classes

II.1. Les fonctions (ou les méthodes de la classe principale)

II.1.1. Créer une fonction retournant une valeur du type « type0 »

II.1.2. Retourner une valeur de la fonction

II.1.3. Créer une fonction retournant aucune valeur

II.1.4. Faire un appel à la fonction

II.2. Les énumérations

II.2.1. Créer une énumération

II.2.2. Affecter une valeur de l'énumération

II.3. Les classes

II.3.1. Visibilités

II.3.2. Les classes de base

II.3.2.1. Créer une classe

II.3.2.2. Définir la classe dans un objet

- II.3.2.3. Ajouter un affichage d'un message de la part de la classe (lors de l'exécution de `System.out.println(monObjet)`):
- II.3.2.4. Créer une méthode (dans une classe)
- II.3.2.5. Faire un appel d'une méthode autre que le constructeur et `toString` ou d'une variable dans une classe
- II.3.2.6. Faire un appel d'une méthode autre que le constructeur et `toString` ou d'une variable en dehors d'une classe
- II.3.2.7. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe
- II.3.3. Les classes internes
- II.3.4. L'héritage
 - II.3.4.1. Créer une classe héritée d'une autre classe
 - II.3.4.2. Définir la classe dans un objet
 - II.3.4.3. Ajouter un affichage d'un message de la part de la classe (lors de l'exécution de `System.out.println(monObjet)`):
 - II.3.4.4. Récupérer le nom de la classe héritée
- II.3.5. Les exceptions
 - II.3.5.1. Exceptions prédéfinies
 - II.3.5.2. Créer une classe d'exception
 - II.3.5.3. Définir une méthode ou une fonction pouvant renvoyer une exception (en dehors de la classe d'exception)
 - II.3.5.4. Provoquer une erreur dans une méthode
 - II.3.5.5. Traiter l'erreur
- II.3.6. Les classes abstraites
 - II.3.6.1. Créer une classe abstraite
 - II.3.6.2. Créer une méthode abstraite
 - II.3.6.3. Créer une classe héritant d'une classe abstraite
 - II.3.6.4. Implémenter une méthode abstraite (dans la classe fille)
 - II.3.6.5. Définir la classe dans un objet
- II.3.7. Les interfaces
 - II.3.7.1. Créer une interface
 - II.3.7.2. Créer une méthode abstraite
 - II.3.7.3. Créer une classe implémentant une ou plusieurs interfaces
 - II.3.7.4. Implémenter une méthode abstraite (dans la classe fille)
 - II.3.7.5. Définir la classe dans un objet
- II.3.8. Les classes anonymes
 - II.3.8.1. Manière simple
 - II.3.8.2. Expression lambda (ne fonctionne que si nous avons 1 seule méthode abstraite)
- II.3.9. Les collections
 - II.3.9.1. Créer une collection
 - II.3.9.2. Définir la collection dans un objet

III. Les instructions

III.1. Instructions de bases

IV. Les packages

V. Les bibliothèques

V.0. Importer une bibliothèque

V.1. `java.util` (`import java.util.*;`)

V.1.1. `Scanner` (`import java.util.Scanner;`)

V.1.2. `List` (`import java.util.List;`)

V.1.3. `Set` (`import java.util.Set;`)

V.1.3. `Map` (`import java.util.List;`)

VI. L'interface graphique (`import javax.swing.*;` et `import java.awt.*;`)

VI.1. Syntaxe

VI.2. Les différents paramètres de la fenêtre (`construteur MaFenetre()`)

VI.3. Gestion de la barre de menu (`creerBarreDeMenu()`)

VI.4. Gestion du panel (`creerPanel()`)

VI.4.1. Les différents layouts

VI.5. Gestion des widgets

VI.5.1. Instructions en commun pour tous les widgets

VI.5.2. Les différents widgets

VI.5.2.1. JLabel (Ajouter du texte)

VI.5.2.2. Image (Ajouter une image avec ImageIcon et Image)

VI.5.2.3. JButton (Ajouter un bouton simple)

VI.5.2.4. JToggleButton (Ajouter un bouton ON/OFF)

VI.5.2.5. JCheckBox (Ajouter une case à cocher)

VI.5.2.6. JRadioButton (Ajouter un bouton radio (sélection unique))

VI.5.2.7. JComboBox (Ajouter une liste d'options)

VI.5.2.8. JList (Ajouter une liste de données)

VI.5.2.9. JTextField (Ajouter une zone de saisie simple)

VI.5.2.10. JTextArea (Ajouter une zone de saisie longue)

VI.5.3. Les différents conteneurs

VI.5.3.1. JPanel (widget de regroupement simple)

VI.5.3.2. JTabbedPane

VI.5.3.3. JScrollPane

VI.6. Gestion du menu déroulant apparaissant au clic droit (JPopupMenu)

VI.7. Les événements (import java.awt.event.*; ou java.awt.AWTEvent.*;)

VI.7.1. Les différents écouteurs d'événements

VI.7.1.1. Écouteurs simples (la classe utilisée est son propre écouteur)

VI.7.1.2. Écouteurs en classe interne

VI.7.1.3. Écouteurs en classe anonyme

VI.7.2. La gestion des événements

VI.7.2.1. ActionListener

VI.7.2.2. ChangeListener

VI.7.2.3. AdjustmentListener

VI.7.2.4. ComponentListener

VI.7.2.5. ContainerListener

VI.7.2.6. FocusListener

VI.7.2.7. ItemListener

VI.7.2.8. KeyListener

VI.7.2.9. DocumentListener

VI.7.2.10. ListSelectionListener

VI.7.2.11. MouseListener

VI.7.2.12. MouseMotionListener

VI.7.2.13. WindowListener

VI.7.2.14. TextListener

VI.8. Les boîtes de dialogue

VI.8.1. MessageDialog (pour afficher des messages avec un bouton OK)

VI.8.2. ConfirmDialog (pour demander une réponse de l'utilisateur parmi des possibilités)

VI.8.3. InputDialog (pour demander une saisie de l'utilisateur)

VI.8.4. OptionDialog (pour choisir ses propres options de réponses)

VI.8.5. FileDialog (pour choisir un fichier)

VI.8.6. JFileChooser (nouvelle méthode pour choisir un fichier)

VI.8.7. JColorChooser (pour choisir une couleur)

VI.9. Les fenêtres modales

I. Bases

I.1. Syntaxe

```
public class NomDuProgramme {  
  
    public static void main(String[] args) {  
        instruction1;  
        instruction2;  
        instruction3;  
    }  
}
```

```
}  
}
```

I.2. Enregistrer et exécuter un programme *NomDuProgramme.java*

Dans la console, tapez la commande suivante :

```
javac *.java
```

puis pour exécuter le programme la commande :

```
java NomDuProgramme
```

Remarque : Dans le cas où vous avez plusieurs fichiers .java, *NomDuProgramme* est le nom de la classe principale où se situe le main.

I.3. Types de variables

I.3.1. Numériques

Type	Type objet	Minimum	Maximum	Description
int (4o)	Integer	-2 147 483 648	2 147 483 647	Nombre entier
short (2o)	Short	-32 768	32 767	Nombre entier
long (8o)	Long	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	Nombre entier
float (4o)	Float	-3.4 _{x10} ^{38f}	3.4 _{x10} ^{38f}	Nombre décimal
double (8o) (2 fois plus précis que float)	Double	-1.7 _{x10} ³⁰⁸	1.1 _{x10} ³⁰⁸	Nombre décimal

I.3.2. Caractères

Type	Type objet	Valeurs possibles	Description
char (1o)	Char	Unicode (65 536 caractères possibles)	Nombre entier ou caractère (1 seul) entre "
String (texte)	String	Infinité de caractères	Chaîne de caractères entre ""

I.3.3. Autres types

Type	Type objet	Description
------	------------	-------------

Object	Object	Type indéfini (tous les autres types héritent de Object)
boolean	Boolean	Valeur pouvant être true ou false

I.4. Commentaires

//Commentaire tenant sur une ligne

```
/*
Commentaire pouvant être sur une ou plusieurs lignes
*/
```

I.5. Opérations

Instruction	Description
1 + 2	Renvoie 3
3 - 1	Renvoie 2
6 * 4	Renvoie 24
5.0 / 2.0	Renvoie 2.5
5 / 2	Renvoie 2 (le quotient sans décimal)
5 % 2	Renvoie 1 (le reste de la division)

I.6. Les variables

Instruction	Description
<i>type variable = valeur;</i>	Déclare une nouvelle variable
int a;	Déclare la variable <i>a</i> comme int
a = 5;	Affecte 5 à une variable
a += 1;	Équivalent à $a = a + 1$;
a -= 1;	Équivalent à $a = a - 1$;
b = a++;	Équivalent à $b = a$; $a = a + 1$;
b = ++a;	Équivalent à $a = a + 1$; $b = a$;
b = a--;	Équivalent à $b = a$; $a = a - 1$;

<code>b = --a;</code>	Équivalent à <code>a = a - 1; b = a;</code>
<code>int a = (int) b;</code>	Convertit le nombre décimal <i>b</i> en nombre entier <i>a</i>
<code>final float pi = 3.14;</code>	Crée une variable constante (non modifiable)
<code>static type variable = valeur;</code>	Déclare une nouvelle variable qui n'est jamais détruite (si déjà exécuté, cette instruction est ignorée)
<code>char caractere = 'c';</code>	Déclare une variable contenant un caractère
<code>String texte = valeur;</code>	Déclare une variable contenant une chaîne de caractère

I.7. Les tableaux

Instruction	Description
<code>type tableau[] = new type[i];</code>	Crée un tableau vide de <i>i</i> éléments
<code>tableau[i]</code>	Renvoie la valeur du tableau à la position <i>i</i> (l'indice de la première valeur est 0)
<code>type tableau[] = {val1, val2...};</code> ou <code>type tableau[] = new type{val1, val2...};</code>	Crée un tableau de <i>i</i> éléments avec des valeurs personnalisées
<code>tableau.length</code>	Renvoie la longueur du tableau

Remarque : Une chaîne de caractère n'est pas un tableau.

I.8. Conditions

Les conditions renvoient true si elle est respectée et false sinon

I.8.1. Opérateurs de comparaison

Condition	Description de ce que vérifie la condition
<code>a == b</code>	<i>a</i> égal à <i>b</i> (seulement en contenu : 1="1")
<code>a === b</code>	<i>a</i> égal à <i>b</i> (en type et en contenu : 1≠"1")

$a < b$	a strictement inférieur à b
$a > b$	a strictement supérieur à b
$a \leq b$	a inférieur ou égal à b
$a \geq b$	a supérieur ou égal à b
$a \neq b$	a n'est pas égal à b (seulement en contenu : $1 \neq "1"$)
$a !== b$	a n'est pas égal à b (en type et en contenu : $1 \neq "1"$)
<code> </code>	À mettre entre deux conditions, permet d'avoir une des deux conditions qui doit être vraie
<code>&&</code>	À mettre entre deux conditions, permet d'avoir deux conditions qui doivent être vraies
<code>!condition</code>	La condition doit être fausse

I.8.2. Tests de conditions

Instruction	Description
<pre>if (condition1) { instruction1; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i>
<pre>if (condition1) { instruction1; } else { instruction2; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i> , sinon, on exécute <i>instruction2</i>
<pre>if (condition1) { instruction1; } else if (condition2) { instruction2; } else { instruction3; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i> , sinon, si <i>condition2</i> est vraie, on exécute <i>instruction2</i> , sinon, on exécute <i>instruction3</i>
<pre>switch (nombre) { case a: instruction1; break; case b: case c: instruction2; break; default: instruction3; }</pre>	Si <i>nombre</i> == <i>a</i> , alors on exécute <i>instruction1</i> , sinon, si <i>nombre</i> == <i>b</i> ou <i>c</i> , on exécute <i>instruction2</i> , sinon, on exécute <i>instruction3</i>

<code>a = (condition1) ? 1 : 0;</code>	Si <i>condition1</i> est vraie, <i>a</i> prend la valeur 1, sinon 0.
--	--

I.9. Boucles

Instruction	Description
<code>for (int i = d; i < f; i++) { instruction1; }</code>	On répète <i>f-d</i> fois <i>instruction1</i> pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris
<code>for (int i = d; i < f; i+=p) { instruction1; }</code>	On répète (<i>f-d</i>)/ <i>p</i> fois <i>instruction1</i> pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris avec pour pas égal à <i>p</i>
<code>for (type elt: tableau) { instruction1; }</code>	On parcourt le tableau (ou une chaîne de caractères) pour <i>elt</i> prenant toutes les valeurs du tableau
<code>while (condition) { instruction1; }</code>	On répète jusqu'à ce que <i>condition</i> soit fausse (peut ne pas être répété)
<code>do { instruction1; } while(condition);</code>	On répète jusqu'à ce que <i>condition</i> soit fausse (est forcément répété une fois)
<code>break;</code>	Permet de sortir d'une boucle sans la terminer (fortement déconseillé)

II. Les méthodes et les classes

II.1. Les fonctions (ou les méthodes de la classe principale)

Les fonctions doivent être écrites juste après la méthode main.

II.1.1. Créer une fonction retournant une valeur du type « *type0* »

```
public static type0 maFonction(type1 variable1, type2 variable2...) {
    instructions;
}
```

II.1.2. Retourner une valeur de la fonction

```
return variable;
```

II.1.3. Créer une fonction retournant aucune valeur

```
public static void maFonction(type1 variable1, type2 variable2...) {
    instructions;
}
```

II.1.4. Faire un appel à la fonction


```
variable = maFonction(valeur1, valeur2...);
```

ou (s'il n'y a pas de variable de retour)

```
maFonction(valeur1, valeur2...);
```

Remarque : il est possible de faire une surcharge de fonctions, c'est-à-dire qu'il est possible de créer deux fonctions identiques avec des paramètres de types différents, ce qui permet à l'interpréteur de choisir la fonction correspondant au type de variables saisies. Idem pour les constructeurs des classes.

II.2. Les énumérations

II.2.1. Créer une énumération

Exemple :

```
public enum JoursSemaine {  
    LUNDI,  
    MARDI,  
    MERCREDI,  
    JEUDI,  
    VENDREDI,  
    SAMEDI,  
    DIMANCHE  
}
```

II.2.2. Affecter une valeur de l'énumération

Exemple :

```
JoursSemaine jour = JoursSemaine.MERCREDI;
```

II.3. Les classes

II.3.1. Visibilités

- **public** : Peut être appelé en dehors de la classe
- **private** : Ne peut être appelé qu'au sein de la classe (pour des valeurs, il est préférable de créer des getter et des setter pour accéder à la valeur depuis l'extérieur)
- **protected** : Ne peut être appelé qu'au sein de la classe et dans les classes héritées

II.3.2. Les classes de base

II.3.2.1. Créer une classe

```
public class MaClasse {  
  
    private type1 variable1;  
    private type2 variable2;  
  
    public static type3 variableStatique3 = valeur;  
  
    visibilite MaClasse(type1 mVariable1, type2 mVariable2...) {  
        this.variable1 = mVariable1;  
        this.variable2 = mVariable2;  
    }  
}
```

Note : Il est possible de surcharger le constructeur *MaClasse* et de créer une variable statique qui reste la même valeur pour tous les objets.

II.3.2.2. Définir la classe dans un objet

```
MaClasse monObjet = new MaClasse(valeur1, valeur2...);
```

II.3.2.3. Ajouter un affichage d'un message de la part de la classe (lors de l'exécution de `System.out.println(monObjet)`):

```
public String toString() {  
    return "message";  
}
```

II.3.2.4. Créer une méthode (dans une classe)

```
visibilite type0 maMethode(type1 variable1, type2 variable2...) {  
    instructions;  
}
```

II.3.2.5. Faire un appel d'une méthode autre que le constructeur et `toString` ou d'une variable dans une classe

```
this.maMethode(valeur1, valeur2...)  
this.variable1 = valeur1;
```

II.3.2.6. Faire un appel d'une méthode autre que le constructeur et `toString` ou d'une variable en dehors d'une classe

```
monObjet.maMethode(valeur1, valeur2...)  
monObjet.variable1 = valeur1;
```

Il est préférable d'utiliser des méthodes (appelées getter et setter) pour modifier les variables non statiques

II.3.2.7. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe

```
MaClasse.maMethode(valeur1, valeur2...)  
MaClasse.variable1 = valeur1;
```

II.3.3. Les classes internes

Les classes internes sont des classes présentes dans une autre classe.

```
public class MaClasse {  
    private class MaClasseInterne {  
    }  
}
```

```
}
```

Il est préférable de mettre la classe interne interne en privée ou en protégée. Si la classe interne est publique, elle est accessible avec *MaClasse.MaClasseInterne*, mais dans ce cas, il est déconseillé d'utiliser une classe interne.

II.3.4. L'héritage

L'héritage permet à des classes filles de reprendre les mêmes caractéristiques que leur classe mère, et d'ajouter des nouveaux attributs et/ou méthodes qui leur sont propres. Il est possible de faire un outrepassement d'une méthode de la classe mère, c'est-à-dire de créer une méthode du même nom qu'une méthode de la classe mère pour la remplacer. L'outrepassement peut être interdit avec le mot-clé « final ». Une classe ne peut hériter que d'une seule classe.

II.3.4.1. Créer une classe héritée d'une autre classe

```
public class MaClasseHeritee extends MaClasse {  
  
    private type4 variable4;  
    private type5 variable5;  
  
    public static type6 variableStatique6 = valeur;  
  
    visibilite MaClasseHeritee(type1 mVariable1, type2 mVariable2..., type4 mVariable4,  
type5 mVariable5...) {  
        super(mVariable1, mVariable2...);  
        this.variable4 = mVariable4;  
        this.variable5 = mVariable5;  
    }  
}
```

La méthode `super()` permet d'appeler le constructeur de la classe initiale.

II.3.4.2. Définir la classe dans un objet

```
MaClasse monObjet = new MaClasseHeritee(valeur1, valeur2..., valeur4, valeur5...);
```

ou

```
MaClasseHeritee monObjet = new MaClasseHeritee(valeur1, valeur2..., valeur4, valeur5...);
```

II.3.4.3. Ajouter un affichage d'un message de la part de la classe (lors de l'exécution de `System.out.println(monObjet)`):

```
public String toString() {  
    return (super.toString() + "message");  
}
```

II.3.4.4. Récupérer le nom de la classe héritée

```
String nomType = monObjet.getClass().getName();
```

II.3.5. Les exceptions

Les classes d'exceptions permettent de gérer les erreurs possibles dans les programmes. Il est possible d'utiliser des exceptions prédéfinies ou d'en créer soi-même.

II.3.5.1. Exceptions prédéfinies

Exception	Description
ClassCastException	Problème de cast
IllegalArgumentException	Problème d'argument
IndexOutOfBoundsException	Sortie du tableau
InputMismatchException	Saisie invalide à l'écran
NullPointerException	Variable nulle

II.3.5.2. Créer une classe d'exception

```
public class MaClasseException extends Exception {  
    public MaClasseException(String message) {  
        super("Erreur : " + message);  
    }  
}
```

II.3.5.3. Définir une méthode ou une fonction pouvant renvoyer une exception (en dehors de la classe d'exception)

```
visibilite type0 maMethode(type1 variable1, type2 variable2...) throws MaClasseException {  
    instructions;  
}
```

II.3.5.4. Provoquer une erreur dans une méthode

```
throw new MaClasseException(message);
```

II.3.5.5. Traiter l'erreur

```
try {  
    instruction1;  
} catch (MaClasseException e) {  
    instruction2;  
} finally {  
    instruction3;  
}
```

Note : finally est facultatif. Il ne sert qu'à présenter clairement ce que fait l'algorithme après avoir passé le try et le catch. La variable e affiche le message d'erreur (qui est entré dans super()). Il est également possible de mettre plusieurs catch pour différencier les différentes exceptions, ou être générale avec Exception. Pour des catch situés dans des méthodes, il est possible de faire throw e; afin de traiter l'erreur plus tard.

II.3.6. Les classes abstraites

Les classes abstraites sont des classes qui ne peuvent pas être instanciées et qui sont utilisées pour définir une structure de classe de base pour les classes dérivées. Les classes abstraites sont déclarées avec le mot-clé `abstract`. Elles peuvent contenir des méthodes abstraites, qui sont des méthodes déclarées sans corps. Les méthodes abstraites sont utilisées pour définir une interface pour les classes dérivées. Les classes dérivées doivent implémenter toutes les méthodes abstraites de la classe abstraite parente.

II.3.6.1. Créer une classe abstraite

```
public abstract class MaClasseAbstraite {  
  
    private type1 variable1;  
    private type2 variable2;  
  
    public static type3 variableStatique3 = valeur;  
  
    visibilite MaClasse(type1 mVariable1, type2 mVariable2...) {  
        this.variable1 = mVariable1;  
        this.variable2 = mVariable2;  
    }  
}
```

II.3.6.2. Créer une méthode abstraite

```
visibilite abstract type0 maMethodeAbstraite(type1 variable1, type2 variable2...);
```

Toutes les méthodes qui ne possèdent pas de mot-clé `abstract` ne doivent pas être obligatoirement implémentées par les classes filles.

II.3.6.3. Créer une classe héritant d'une classe abstraite

```
public class MaClasseHeritee extends MaClasseAbstraite {  
  
    private type4 variable4;  
    private type5 variable5;  
  
    public static type6 variableStatique6 = valeur;  
  
    visibilite MaClasseHeritee(type1 mVariable1, type2 mVariable2..., type4 mVariable4,  
type5 mVariable5...) {  
        super(mVariable1, mVariable2...);  
        this.variable4 = mVariable4;  
        this.variable5 = mVariable5;  
    }  
}
```

II.3.6.4. Implémenter une méthode abstraite (dans la classe fille)

```
@Override  
visibilite type0 maMethodeAbstraite(type1 variable1, type2 variable2...) {  
    instructions;  
}
```

Le mot-clé `@Override` permet d'indiquer à l'interpréteur qu'il s'agit d'une méthode abstraite implémentée, et à détecter des erreurs éventuelles.

II.3.6.5. Définir la classe dans un objet

```
MaClasseAbstraite monObjet = new MaClasseHeritee(valeur1, valeur2...);
```

ou

```
MaClasseHeritee monObjet = new MaClasseHeritee(valeur1, valeur2...);
```

Il est possible de définir directement la classe abstraite dans un objet, sans faire héritage. Voir les classes anonymes.

II.3.7. Les interfaces

Les interfaces sont des classes abstraites qui permettent de définir un ensemble de méthodes sans les implémenter. Elles sont très utiles pour réduire la dépendance entre classes. Les classes peuvent implémenter plusieurs interfaces et doivent fournir une implémentation pour chacune des méthodes annoncées.

II.3.7.1. Créer une interface

```
public interface MonInterface {  
  
}
```

II.3.7.2. Créer une méthode abstraite

```
visibilite type0 maMethodeAbstraite(type1 variable1, type2 variable2...);
```

Toutes les méthodes n'ont pas besoin de mot-clé `abstract` car toutes les méthodes d'une interface doivent être obligatoirement implémentées par les classes filles.

II.3.7.3. Créer une classe implémentant une ou plusieurs interfaces

```
public class MaClasseHeritee extends MaClasseAbstraite implements MonInterface1,  
MonInterface2... {  
  
    private type4 variable4;  
    private type5 variable5;  
  
    public static type6 variableStatique6 = valeur;  
  
    visibilite MaClasseHeritee(type1 mVariable1, type2 mVariable2..., type4 mVariable4,  
type5 mVariable5...) {  
        super(mVariable1, mVariable2...);  
        this.variable4 = mVariable4;  
        this.variable5 = mVariable5;  
    }  
}
```

II.3.7.4. Implémenter une méthode abstraite (dans la classe fille)

```
@Override
visibilite type0 maMethodeAbstraite(type1 variable1, type2 variable2...) {
    instructions;
}
```

Le mot-clé `@Override` permet d'indiquer à l'interpréteur qu'il s'agit d'une méthode abstraite implémentée, et à détecter des erreurs éventuelles.

II.3.7.5. Définir la classe dans un objet

```
MonInterface monObjet = new MaClasseAbstraite(valeur1, valeur2...);
```

ou

```
MaClasseAbstraite monObjet = new MaClasseAbstraite(valeur1, valeur2...);
```

ou

```
MaClasse monObjet = new MaClasseAbstraite(valeur1, valeur2...);
```

Il est possible de définir directement l'interface dans un objet, sans faire d'implémentation (uniquement s'il n'y a qu'une seule interface). Voir les classes anonymes.

II.3.8. Les classes anonymes

Les classes anonymes sont généralement des classes abstraites ou des interfaces définies directement dans un objet sans avoir à passer par une classe héritant d'une classe abstraite (ça peut également être une classe simple, mais c'est déconseillé). Chaque classe anonyme doit implémenter obligatoirement toutes les méthodes abstraites entre `{}`. Elle peut également contenir des méthodes non obligatoires à implémenter. Cela peut être utile si le code ne peut être exécuté que par un seul objet.

II.3.8.1. Manière simple

```
MaClasseAbstraite monObjet = new MaClasseAbstraite(valeur1, valeur2...) {
};
```

II.3.8.2. Expression lambda (ne fonctionne que si nous avons 1 seule méthode abstraite)

```
MaClasseAbstraite monObjet = (valeur1, valeur2...) -> {
    instructions;
};
```

Pas besoin d'entrer le nom de la méthode abstraite. Seules les paramètres sont renseignées.

II.3.9. Les collections

Les collections sont des classes qui peuvent avoir un ou plusieurs type de données personnalisables.

II.3.9.1. Créer une collection

```
public class MaCollection<T1, T2...> {
```

```

private type1 variable1;
private type2 variable2;
private T1 variable3;
private T2 variable4;

public static type3 variableStatique3 = valeur;

    visibilite MaClasse(type1 mVariable1, type2 mVariable2..., T1 mVariable3, T2
mVariable4...) {
        this.variable1 = mVariable1;
        this.variable2 = mVariable2;
        this.variable3 = mVariable3;
        this.variable4 = mVariable4;
    }
}

```

Il est possible d'utiliser `<T1 extend Comparable<T1>>` afin d'utiliser la méthode `compareTo()` entre 2 objets qui renvoie un int.

II.3.9.2. Définir la collection dans un objet

```

MaCollection<TypeObjet1, TypeObjet2...> monObjet = new maCollection<>(valeur1, valeur2...,
valeur4, valeur5...);

```

III. Les instructions

III.1. Instructions de bases

Instruction	Description
<code>System.out.print("texte");</code>	Affiche un texte dans la console
<code>System.out.println("texte");</code>	Affiche un texte dans la console avec un retour à la ligne (peut être remplacé par <code>\n</code> , placé directement dans la chaîne) (pour une tabulation, entrez <code>\t</code> ; <code>\b</code> : retour en arriere ; <code>\f</code> : nouvelle page)
<code>System.out.println(variable);</code> <code>System.out.println("Valeur : " + variable);</code>	Affiche une variable (ici <i>variable</i>) dans la console
<code>variable.getClass().getName();</code>	Renvoie le nom de la classe de la variable
<code>chaîne1.equals(chaîne2);</code>	Vérifie que la première chaîne est égale à la deuxième (fonctionne également avec les objets)
<code>chaîne1.concat(chaîne2);</code>	Concatène une chaîne avec une autre
<code>chaîne.toUpperCase();</code>	Renvoie la chaîne de caractères en majuscule
<code>chaîne.toLowerCase();</code>	Renvoie la chaîne de caractères en minuscule
<code>chaîne.length();</code>	Renvoie la longueur de la chaîne

<code>chaine.trim();</code>	Supprime tous les espaces avant et après la chaîne
<code>chaine.replace(ancien, nouveau);</code>	Remplace tous les caractères par le nouveau
<code>chaine.charAt(i);</code>	

IV. Les packages

V. Les bibliothèques

V.0. Importer une bibliothèque

```
import bibliotheque;
```

V.1. java.util (import java.util.*;)

V.1.1. Scanner (import java.util.Scanner;)

Scanner permet de lire des données entrées dans la console.

Dans la classe où vous souhaitez l'utiliser, ajoutez avant le constructeur :

```
private static Scanner entree = new Scanner(System.in);
```

Instruction	Utilité
<code>variable = entree.nextLine();</code>	Demande une valeur de type String avec le retour dans <i>variable</i>
<code>variable = entree.nextInt();</code> <code>entree.nextLine();</code>	Demande une valeur de type int de manière sécurisée avec le retour dans <i>variable</i>

V.1.2. List (import java.util.List;)

List permet de créer des tableaux de valeurs d'une taille variable (c'est-à-dire des listes) et de pouvoir faire diverses manipulations facilement.

Les différentes listes possibles :

- ArrayList (import java.util.ArrayList;) : Plus rapide pour les opérations de recherche (il reprend le principe d'une liste chaînée)
- LinkedList (import java.util.LinkedList;) : Plus rapide pour les opérations d'ajout et de suppression (il reprend le principe d'un tableau)

Instruction	Utilité
<code>List<typeObjet> liste = new XXXList<>();</code>	Crée une liste vide

<code>Liste.get(i);</code>	Renvoie la valeur à la position <i>i</i>
<code>Liste.add(valeur);</code>	Ajoute une valeur dans la liste
<code>Liste.remove(i);</code>	Enlève la valeur de la liste à la position <i>i</i>
<code>Liste.remove(valeur);</code>	Enlève la valeur de la liste qui contient l'objet recherché
<code>Liste.size();</code>	Renvoie la longueur de la liste
<code>List<typeObjet> nouvelleListe = Liste.clone();</code>	Crée une copie de la liste
<code>Liste.isEmpty();</code>	Vérifie si la liste est vide
<code>Liste.indexOf(valeur);</code>	Renvoie la position de la valeur recherchée ou -1 si elle n'est pas trouvée
<code>Liste.contains(valeur);</code>	Vérifie si la liste contient la valeur recherchée

V.1.3. Set (import java.util.Set;)

Set permet de créer des listes chaînées de valeurs obligatoirement différentes d'une taille variable (c'est-à-dire des listes) et de pouvoir faire diverses manipulations facilement.

Les différentes listes possibles :

- HashSet (import java.util.HashSet;) : Plus rapide pour l'accès et la manipulation des éléments (il reprend le principe du hachage)
- TreeSet (import java.util.TreeSet;) : Plus rapide pour trier les éléments (il reprend le principe d'un arbre)

Instruction	Utilité
<code>Set<typeObjet> Liste = new XXXSet<>();</code>	Crée une collection vide
<code>Liste.add(valeur);</code>	Ajoute une valeur dans la collection
<code>Liste.remove(valeur);</code>	Enlève la valeur de la collection (renvoie false si elle n'a pas été trouvée)
<code>Liste.size();</code>	Renvoie la longueur de la collection
<code>Iterator iterateur = Liste.iterator(); while (iterateur.hasNext()) { typeObjet element = (typeObjet)iterateur.next(); }</code>	Permet de parcourir la collection

V.1.3. Map (import java.util.List;)

Map permet de créer des dictionnaires de valeurs d'une taille variable avec des valeurs associées à des clés.

Les différents dictionnaires possibles :

- HashMap (import java.util.HashMap;) : Plus rapide pour les opérations de recherche (il reprend le principe du hachage)
- TreeMap (import java.util.TreeMap;) : Éléments triés (il reprend le principe d'un arbre)

Instruction	Utilité
<code>Map<typeObjet1, typeObjet2> dictionnaire = new XXXMap<>();</code>	Crée un dictionnaire vide
<code>dictionnaire.get(cle);</code>	Renvoie la valeur associée à la clé
<code>dictionnaire.put(cle, valeur);</code>	Ajoute une valeur dans le dictionnaire

VI. L'interface graphique (import javax.swing.*; et import java.awt.*;)

Chaque fenêtre a son fichier java. La fenêtre principale n'est pas le programme principal. Elle sert à initialiser et ouvrir la fenêtre principale.

VI.1. Syntaxe

```
import javax.swing.*;
import java.awt.*;

public class MaFenetre extends JFrame {

    public MaFenetre() {
        /* Instructions pour paramétrer la fenêtre */

        this.setJMenuBar(this.creerBarreDeMenu());
        this.setContentPane(this.creerPanel());
    }

    JMenuBar maBarreDeMenu;

    private JMenuBar creerBarreDeMenu() {
        maBarreDeMenu = new JMenuBar();

        /* Instructions pour créer des menus */

        return maBarreDeMenu;
    }

    JPanel monPanel;

    private JPanel creerPanel() {
        monPanel = (JPanel) getContentPane();

        /* Instructions pour créer des widgets */

        return monPanel;
    }
}
```

VI.2. Les différents paramètres de la fenêtre (constructeur *MaFenetre()*)

Instruction	Description
<code>UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());</code>	Mettre le style du système d'exploitation au lieu du style de java par défaut (peut provoquer une erreur si le style est inexistant)
<code>this.setTitle("nom de la fenêtre");</code>	Changer le nom de la fenêtre (en haut à gauche)
<code>this.setIconImage(new ImageIcon("src/icone.png").getImage());</code>	Changer l'icône de l'application (en haut à gauche)
<code>this.setSize(Longueur, hauteur);</code>	Redimensionner la fenêtre
<code>this.setResizable(false);</code>	Interdire le redimensionnement de la fenêtre
<code>this.setExtendedState(JFrame.MAXIMIZED_BOTH);</code>	Ouvrir la fenêtre en plein écran
<code>this.setLocationRelativeTo(null);</code>	Placer la fenêtre au centre de l'écran
<code>this.setDefaultCloseOperation(actionFermeture);</code>	Changer l'action au clic sur la croix : <ul style="list-style-type: none">• <code>JFrame.EXIT_ON_CLOSE</code> : Fermer l'application• <code>JFrame.HIDE_ON_CLOSE</code> : Fermer la fenêtre sans quitter l'application• <code>JFrame.DISPOSE_ON_CLOSE</code> : Fermer la fenêtre actuelle et rend la main à la fenêtre parent• <code>JFrame.DO_NOTHING_ON_CLOSE</code> : Bloquer la fermeture de la fenêtre
<code>this.pack();</code>	Adapter la taille de la fenêtre à ses composants
<code>this.setVisible(true);</code>	Afficher la fenêtre (ouvre la fenêtre au moment où elle est appelée)

VI.3. Gestion de la barre de menu (*creerBarreDeMenu()*)

Instruction	Description
<code>JMenu menu1 = new JMenu("mon menu"); maBarreDeMenu.add(menu1);</code>	Créer un menu
<code>JMenuItem option1 = new JMenuItem("mon option"); menu1.add(option1);</code>	Créer un item d'un menu
<code>JMenuItem option1 = new JMenuItem("mon option", new ImageIcon("src/icone.png"));</code>	Créer un item d'un menu avec une icône

<code>menu1.add(option1);</code>	
<code>JCheckBoxMenuItem option2 = new JCheckBoxMenuItem("mon option"); menu1.add(option2);</code>	Créer un item à cocher d'un menu
<code>JRadioButtonMenuItem option3 = new JRadioButtonMenuItem("mon option"); menu1.add(option3);</code>	Créer un item radio d'un menu
<code>menu1.addSeparator();</code>	Ajouter un trait de séparation
<code>menu1.setIcon(new ImageIcon("src/icone.png").getImage());</code>	Ajouter une icône

Pour que chaque items puissent effectuer une action, il faut leur ajouter un écouteur. Voir les événements.

VI.4. Gestion du panel (*creerPanel()*)

VI.4.1. Les différents layouts

Instruction	Insertion de widgets	Description
<code>monPanel.setLayout(new FlowLayout());</code> <code>monPanel.setLayout(new FlowLayout(FlowLayout.LEFT));</code> <code>monPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));</code> <code>monPanel.setLayout(new FlowLayout(FlowLayout.CENTER, espacementH, espacementV));</code>	<code>monPanel.add(widget);</code>	Place les composants les uns à la suite des autres de gauche à droite et de façon centrée par défaut, en passant à la ligne suivante si nécessaire (avec un espacement horizontal et un espacement vertical si les paramètres sont spécifiés)
<code>monPanel.setLayout(new GridLayout(nbLignes, nbColonnes));</code> <code>monPanel.setLayout(new GridLayout(nbLignes, nbColonnes, espacementH, espacementV));</code>	<code>monPanel.add(widget, ligne, colonne);</code>	Place les composants dans une grille, soit dans une case spécifiée, soit dans la prochaine case vide si rien n'est renseigné (avec un espacement horizontal et un espacement vertical si les paramètres sont spécifiés)
<code>monPanel.setLayout(new BorderLayout());</code> <code>monPanel.setLayout(new BorderLayout(espacementH, espacementV));</code>	<code>monPanel.add(widget, BorderLayout.CENTER);</code> <code>monPanel.add(widget, BorderLayout.NORTH);</code> <code>monPanel.add(widget, BorderLayout.SOUTH);</code> <code>monPanel.add(widget,</code>	Découpe l'écran en 5 régions (avec un espacement horizontal et un espacement vertical si les paramètres sont spécifiés)

	<code>BorderLayout.WEST);</code> <code>monPanel.add(widget,</code> <code>BorderLayout.EAST);</code>	
<code>monPanel.setLayout(null);</code>	<code>widget.setBounds(x, y,</code> <code>largeur, hauteur);</code>	Place les widgets de manière absolue grâce à des valeurs personnalisées (fortement déconseillé)

VI.5. Gestion des widgets

VI.5.1. Instructions en commun pour tous les widgets

Instruction	Description
<code>widget.setLocation(x, y);</code>	Changer les coordonnées du widget à partir du bord en haut à gauche
<code>widget.setWidth(largeur);</code> <code>widget.setHeight(hauteur);</code>	Changer la taille du widget
<code>widget.setBackground(new Color (r, g, b));</code>	Changer la couleur du fond
<code>widget.setBorder(</code> <code>BorderFactory.createLineBorder(</code> <code>new Color(r, g, b)</code> <code>),</code> <code>epaisseur</code> <code>);</code>	Changer la bordure
<code>widget.setForeground(new Color (r, g, b));</code>	Changer la couleur du texte
<code>widget.setOpaque(booleen);</code>	Rendre visible ou invisible l'arrière-plan
<code>widget.setVisible(booleen);</code>	Rendre visible ou invisible le widget
<code>int x = widget.getX();</code> <code>int y = widget.getY();</code>	Récupérer les coordonnées du widget à partir du bord en haut à gauche
<code>int largeur = widget.getWidth();</code> <code>int hauteur = widget.getHeight();</code>	Récupérer la taille du widget

VI.5.2. Les différents widgets

VI.5.2.1. JLabel (Ajouter du texte)

Instruction	Description
-------------	-------------

<code>Label0 = new JLabel("Mon texte");</code> <code>Label0 = new JLabel("Mon texte", SwingConstants.CENTER);</code>	Créer le widget (doit être initialisé dans la classe)
<code>Label0.setFont(new Font("Calibri", Font.PLAIN, <i>taille</i>));</code>	Changer le style et la taille du texte (Normal : Font.PLAIN ; Italique : Font.ITALIC ; Gras : Font.BOLD)
<code>Label0.setText("message");</code>	Changer le texte
<code>String monTexte = Label0.getText();</code>	Récupérer le texte

VI.5.2.2. Image (Ajouter une image avec ImageIcon et Image)

Instruction	Description
<code>ImageIcon image0 = new ImageIcon("src/image.png");</code>	Ouvrir une image
<code>Image image0AModifier = image0.getImage();</code>	Permettre de faire les modifications de l'image grâce aux instructions ci-dessous
<code>image0AModifier = image0AModifier.getScaledInstance(<i>largeur</i>, <i>hauteur</i>, Image.SCALE_SMOOTH);</code>	Changer la taille de l'image
<code>image0 = new ImageIcon(image0AModifier);</code>	Appliquer les modifications sur l'image ouverte (sans écraser le fichier)
<code>label1 = new JLabel(image0);</code>	Créer le widget contenant l'image (doit être initialisé dans la classe)

VI.5.2.3. JButton (Ajouter un bouton simple)

Instruction	Description
<code>bouton0 = new JButton("Mon bouton");</code>	Créer le widget (doit être initialisé dans la classe)
<code>bouton0.setEnabled(<i>booléen</i>);</code>	Débloquer ou bloquer le bouton
<code>bouton0.setFont(new Font("Calibri", Font.PLAIN, <i>taille</i>));</code>	Changer le style et la taille du texte (Normal : Font.PLAIN ; Italique : Font.ITALIC ; Gras : Font.BOLD)
<code>bouton0.setText("message");</code>	Changer le texte

<code>String monTexte = bouton0.getText();</code>	Récupérer le texte
---	--------------------

VI.5.2.4. JToggleButton (Ajouter un bouton ON/OFF)

Instruction	Description
<code>bouton1 = new JToggleButton("Mon interrupteur", boolean);</code>	Créer le widget (doit être initialisé dans la classe)
<code>bouton1.setEnabled(boolean);</code>	Débloquer ou bloquer le bouton
<code>bouton1.setFont(new Font("Calibri", Font.PLAIN, taille));</code>	Changer le style et la taille du texte (Normal : Font.PLAIN ; Italique : Font.ITALIC ; Gras : Font.BOLD)
<code>bouton1.setSelected(boolean);</code>	Activer ou désactiver le bouton
<code>bouton1.setText("message");</code>	Changer le texte
<code>String monTexte = bouton1.getText();</code>	Récupérer le texte
<code>ButtonGroup groupeBoutons0 = new ButtonGroup();</code>	Créer un groupe de boutons (pour avoir une sélection unique)
<code>groupeBoutons0.add(bouton1);</code> <code>groupeBoutons0.add(bouton2);</code>	Ajouter un bouton dans un groupe

VI.5.2.5. JCheckbox (Ajouter une case à cocher)

Instruction	Description
<code>box1 = new JCheckBox("Cocher ici");</code> <code>box2 = new JCheckBox("Cocher également ici");</code>	Créer le widget (doit être initialisé dans la classe)
<code>box1.setFont(new Font("Calibri", Font.PLAIN, taille));</code>	Changer le style et la taille du texte (Normal : Font.PLAIN ; Italique : Font.ITALIC ; Gras : Font.BOLD)
<code>box1.setText("message");</code>	Changer le texte
<code>String monTexte = box1.getText();</code>	Récupérer le texte
<code>box1.isSelected();</code>	Vérifier si la case est sélectionnée
<code>box1.setSelected(boolean);</code>	Sélectionner ou désélectionner la case

VI.5.2.6. JRadioButton (Ajouter un bouton radio (sélection unique))

Instruction	Description
<code>radio1 = new JRadioButton("Cocher ici"); radio2 = new JRadioButton("Cocher plutôt ici");</code>	Créer le widget (doit être initialisé dans la classe)
<code>radio1.setFont(new Font("Calibri", Font.PLAIN, <i>taille</i>));</code>	Changer le style et la taille du texte (Normal : Font.PLAIN ; Italique : Font.ITALIC ; Gras : Font.BOLD)
<code>radio1.setText("message");</code>	Changer le texte
<code>String monTexte = radio1.getText();</code>	Récupérer le texte
<code>radio1.isSelected();</code>	Vérifier si le bouton radio est sélectionné
<code>radio1.setSelected(<i>booléen</i>);</code>	Sélectionner ou désélectionner le bouton radio
<code>ButtonGroup groupeRadios0 = new ButtonGroup();</code>	Créer un groupe de boutons radios (pour avoir une sélection unique)
<code>groupeRadios0.add(radio1); groupeRadios0.add(radio2);</code>	Ajouter une radio dans un groupe

VI.5.2.7. JComboBox (Ajouter une liste d'options)

Instruction	Description

VI.5.2.8. JList (Ajouter une liste de données)

Instruction	Description

VI.5.2.9. JTextField (Ajouter une zone de saisie simple)

Instruction	Description

VI.5.2.10. JTextArea (Ajouter une zone de saisie longue)

Instruction	Description

VI.5.3. Les différents conteneurs

Les conteneurs sont des widgets qui peuvent regrouper plusieurs autres widgets.

VI.5.3.1. JPanel (widget de regroupement simple)

Instruction	Description
<code>panel1 = new JPanel();</code>	Créer le widget (doit être initialisé dans la classe)
<code>panel1.setBorder(BorderFactory.createLineBorder(new Color (r, g, b)), epaisseur);</code>	Ajouter une bordure simple au panneau
<code>panel1.setBorder(BorderFactory.createTitleBorder("titreA"));</code>	Ajouter une bordure titrée au panneau

VI.5.3.2. JTabbedPane

Instruction	Description

VI.5.3.3. JScrollPane

Instruction	Description

VI.6. Gestion du menu déroulant apparaissant au clic droit (JPopupMenu)

Instruction	Description
<code>JPopupMenu pop = new JMenu("mon menu"); widget.setComponentPopupMenu(pop);</code>	Créer un menu
<code>JMenuItem option1 = new JMenuItem("mon option"); pop.add(option1);</code>	Créer un item d'un menu
<code>JMenuItem option1 = new JMenuItem("mon option", new ImageIcon("src/icone.png")); pop.add(option1);</code>	Créer un item d'un menu avec une icône

<code>JCheckBoxMenuItem option2 = new JCheckBoxMenuItem("mon option"); pop.add(option2);</code>	Créer un item à cocher d'un menu
<code>JRadioButtonMenuItem option3 = new JRadioButtonMenuItem("mon option"); pop.add(option3);</code>	Créer un item radio d'un menu
<code>pop.addSeparator();</code>	Ajouter un trait de séparation

Pour que chaque items puissent effectuer une action, il faut leur ajouter un écouteur (pas besoin de l'ajouter sur le menu déroulant). Voir les événements.

VI.7. Les événements (import java.awt.event.*; ou java.awt.AWTEvent.*;)

VI.7.1. Les différents écouteurs d'événements

VI.7.1.1. Écouteurs simples (la classe utilisée est son propre écouteur)

```
widget.addXXXListener(this);
```

Note : l'implémentation des méthodes doit se faire dans la classe dans lequel est placé l'écouteur. Il faut également ajouter implements XXXListener à la classe utilisée.

VI.7.1.2. Écouteurs en classe interne

```
widget.addXXXListener(new MonEcouteurXXX());
```

Note : Il faudra créer une classe appelée *MonEcouteurXXX* et y ajouter implements XXXListener.

VI.7.1.3. Écouteurs en classe anonyme

```
widget.addXXXListener(new XXXListener() {  
    /* implémentationDesMéthodes */  
});
```

VI.7.2. La gestion des événements

Chaque événement utilise une méthode d'enregistrement, qui nécessite d'implémenter des méthodes qui exécute des instructions selon l'événement.

Chacune de ces méthode doit être précédée du mot-clé : `@Override`, qui indique à l'interpréteur que l'on surcharge ou implémente une méthode correctement.

Pour récupérer le composant source, il est possible d'utiliser `e.getSource()`

VI.7.2.1. ActionListener

ActionListener est l'écouteur le plus utilisé pour des événements simples comme des clics sur des boutons, touche Entrée dans une zone de saisie, etc.

Composants concernés :

- JButton
- JRadioButton
- JCheckBox
- JComboBox
- JTextField

Méthode d'enregistrement : `addActionListener()`

Méthodes à implémenter	Exécution
<code>actionPerformed(ActionEvent e)</code>	Après un clic, la touche Entrée...

VI.7.2.2. ChangeListener

ChangeListener est l'écouteur le plus souvent utilisé détecter des changements.

Composants concernés :

- JRadioButton
- JCheckBox
- JComboBox

Méthode d'enregistrement : `addChangeListener()`

Méthodes à implémenter	Exécution
<code>stateChanged(ChangeEvent e)</code>	Dès le changement d'état du composant

VI.7.2.3. AdjustmentListener

AdjustmentListener détecte les ajustements d'une barre de défilement.

Composants concernés :

- JScrollBar

Méthode d'enregistrement : `addAdjustmentListener()`

Méthodes à implémenter	Exécution
<code>adjustmentValueChanged(AdjustmentEvent e)</code>	Dès le changement du niveau de la barre de défilement

VI.7.2.4. ComponentListener

ComponentListener détecte les déplacements, l'affichage, le masquage ou la modification de la taille d'un composant (par exemple une fenêtre).

Composants concernés :

- Component (tous)

Méthode d'enregistrement : `addComponentListener()`

Méthodes à implémenter	Exécution
<code>componentHidden(ComponentEvent e)</code>	Dès que le composant a été caché
<code>componentMoved(ComponentEvent e)</code>	Dès que le composant a été déplacé
<code>componentResized(ComponentEvent e)</code>	Dès que le composant a été redimensionné
<code>componentShown(ComponentEvent e)</code>	Dès que le composant a été affiché

VI.7.2.5. ContainerListener

ContainerListener détecte des ajouts ou des suppressions des composants dans un conteneur.

Composants concernés :

- Component (tous les conteneurs)

Méthode d'enregistrement : `addContainerListener()`

Méthodes à implémenter	Exécution
<code>componentAdded(ContainerEvent e)</code>	Dès l'ajout d'un composant dans le conteneur
<code>componentRemoved(ContainerEvent e)</code>	Dès la suppression d'un composant dans le conteneur

VI.7.2.6. FocusListener

FocusListener détecte des survols de la souris sur des composants.

Composants concernés :

- Component (tous)

Méthode d'enregistrement : `addFocusListener()`

Méthodes à implémenter	Exécution
<code>focusGained(FocusEvent e)</code>	Dès le survol de la souris
<code>focusLost(FocusEvent e)</code>	Dès la fin du survol de la souris

VI.7.2.7. ItemListener

ItemListener détecte les éléments sélectionnés dans une liste de possibilités.

Composants concernés :

- JCheckBox
- JComboBox
- List (Attention : Pas JList)
- JRadioButton

Méthode d'enregistrement : `addItemListener()`

Méthodes à implémenter	Exécution
<code>itemStateChanged(ItemEvent e)</code>	Dès la sélection ou la désélection d'un élément

VI.7.2.8. KeyListener

KeyListener détecte les actions sur les touches du clavier.

Composants concernés :

- Component (tous)

Méthode d'enregistrement : `addKeyListener()`

Récupérer la touche sélectionnée (renvoie un int) : `e.getKeyCode()`

Méthodes à implémenter	Exécution
<code>keyTyped(KeyEvent e)</code>	Dès l'appui sur une touche
<code>keyPressed(KeyEvent e)</code>	Dès le maintien sur une touche
<code>keyReleased(KeyEvent e)</code>	Dès le relâchement d'une touche

Note : Au lieu d'implémenter KeyListener, il est possible d'hériter de KeyAdapter afin de ne pas être obligé d'implémenter toutes les méthodes.

VI.7.2.9. DocumentListener

DocumentListener détecte les modifications d'un document

Composants concernés :

- Document

Méthode d'enregistrement : `addDocumentListener()`

Méthodes à implémenter	Exécution
<code>changedUpdate(DocumentEvent e)</code>	Dès qu'un attribut ou un ensemble d'attributs a été modifié

<code>insertUpdate(DocumentEvent e)</code>	Dès qu'il y a eu une insertion dans le document
<code>removeUpdate(DocumentEvent e)</code>	Dès qu'une partie du document a été supprimée

VI.7.2.10. ListSelectionListener

ListSelectionListener détecte le changement de la sélection d'une liste.

Composants concernés :

- JList
- JTable

Méthode d'enregistrement : `addListSelectionListener()`

Méthodes à implémenter	Exécution
<code>valueChanged(ListSelectionEvent e)</code>	Dès qu'une valeur a été changée

VI.7.2.11. MouseListener

MouseListener détecte les actions sur la souris.

Composants concernés :

- Component (tous)

Méthode d'enregistrement : `addMouseListener()`

Méthodes à implémenter	Exécution
<code>mouseClicked(MouseEvent e)</code>	Dès que le bouton de la souris a été cliqué (enfoncé et relâché) sur un composant
<code>mouseEntered(MouseEvent e)</code>	Dès que la souris entre dans un composant
<code>mouseExited(MouseEvent e)</code>	Dès que la souris quitte un composant
<code>mousePressed(MouseEvent e)</code>	Dès qu'un bouton de la souris a été enfoncé sur un composant
<code>mouseReleased(MouseEvent e)</code>	Dès qu'un bouton de la souris a été relâché sur un composant

Note : Au lieu d'implémenter MouseListener, il est possible d'hériter de MouseAdapter afin de ne pas être obligé d'implémenter toutes les méthodes.

VI.7.2.12. MouseMotionListener

MouseMotionListener détecte les actions de la souris sur un composant.

Composants concernés :

- Component (tous)

Méthode d'enregistrement : `addMouseMotionListener()`

Méthodes à implémenter	Exécution
<code>mouseDragged(MouseEvent e)</code>	Dès qu'un bouton de la souris est enfoncé sur un composant puis déplacé
<code>mouseMoved(MouseEvent e)</code>	Dès que le curseur de la souris a été déplacé sur un composant mais qu'aucun bouton n'a été enfoncé

Note : Au lieu d'implémenter MouseMotionListener, il est possible d'hériter de MouseMotionAdapter afin de ne pas être obligé d'implémenter toutes les méthodes.

VI.7.2.13. WindowListener

WindowListener détecte les actions de la fenêtre.

Composants concernés :

- Window

Méthode d'enregistrement : `addWindowListener()`

Méthodes à implémenter	Exécution
<code>windowActivated(WindowEvent e)</code>	Dès que la fenêtre est définie comme étant la fenêtre active
<code>windowClosed(WindowEvent e)</code>	Dès qu'une fenêtre a été fermée
<code>windowClosing(WindowEvent e)</code>	Dès que l'utilisateur tente de fermer la fenêtre à partir du menu système de la fenêtre
<code>windowDeactivated(WindowEvent e)</code>	Dès qu'une fenêtre n'est plus la fenêtre active
<code>windowDeiconified(WindowEvent e)</code>	Dès qu'une fenêtre passe d'un état réduit à un état normal
<code>windowIconified(WindowEvent e)</code>	Dès qu'une fenêtre passe d'un état normal à un état réduit
<code>windowOpened(WindowEvent e)</code>	Dès la première ouverture

Note : Au lieu d'implémenter WindowListener, il est possible d'hériter de WindowAdapter afin de ne pas être obligé d'implémenter toutes les méthodes.

VI.7.2.14. TextListener

TextListener détecte les actions sur une zone de texte.

Composants concernés :

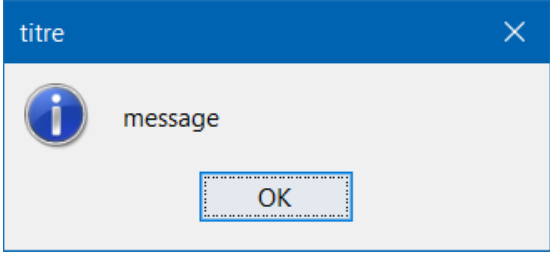
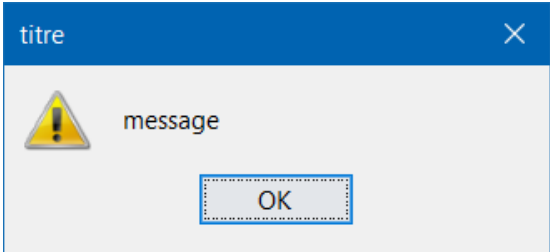
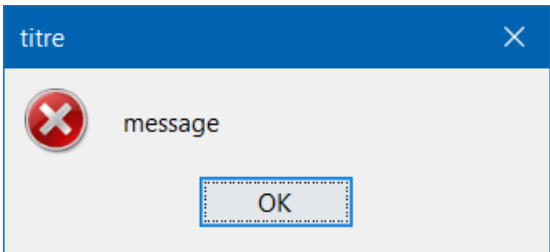
- JTextField

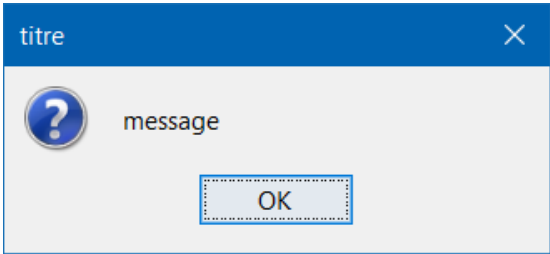
Méthode d'enregistrement : addTextListener()

Méthodes à implémenter	Exécution
textValueChanged(TextEvent e)	Dès que la valeur a changée

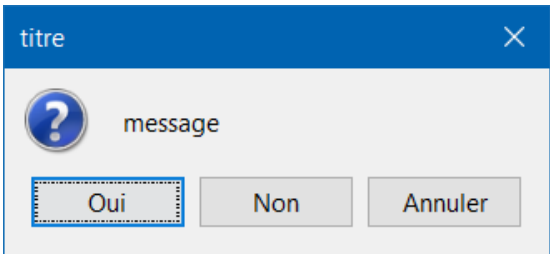
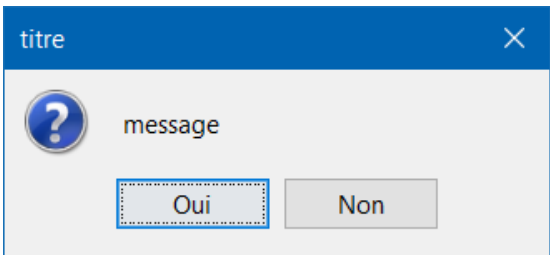
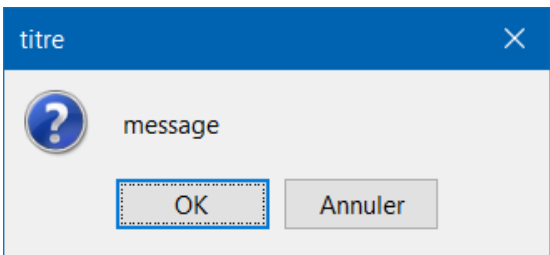
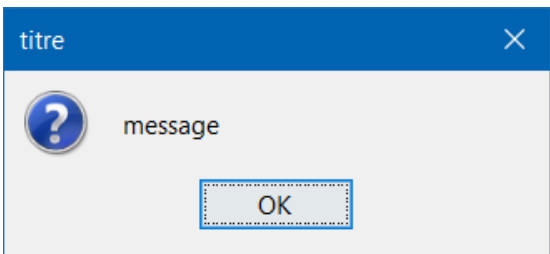
VI.8. Les boîtes de dialogue

VI.8.1. MessageDialog (pour afficher des messages avec un bouton OK)

Instruction	Résultat
<code>JOptionPane.showMessageDialog(fenetre, "message", "titre", JOptionPane.INFORMATION_MESSAGE);</code>	
<code>JOptionPane.showMessageDialog(fenetre, "message", "titre", JOptionPane.WARNING_MESSAGE);</code>	
<code>JOptionPane.showMessageDialog(fenetre, "message", "titre", JOptionPane.ERROR_MESSAGE);</code>	

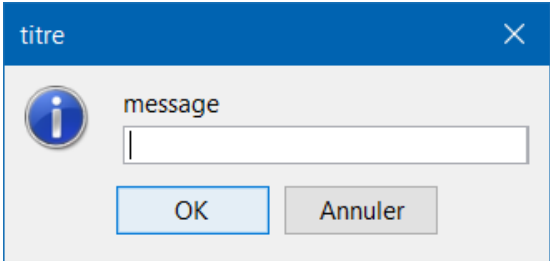
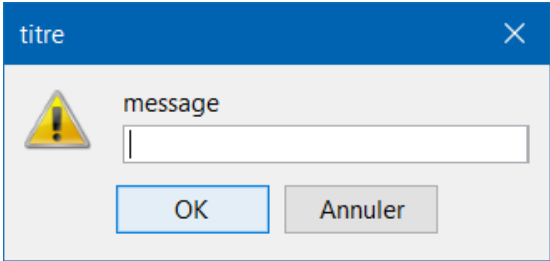
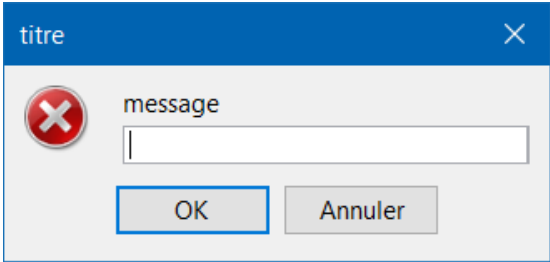
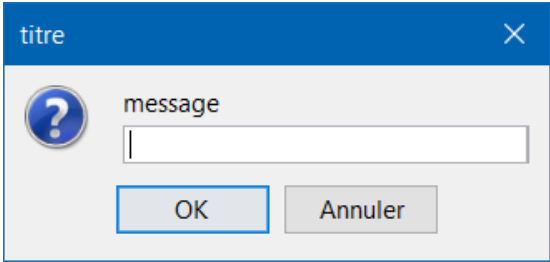
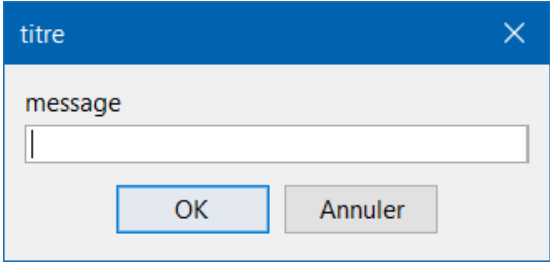
<pre>JOptionPane.showMessageDialog(<i>fenetre</i>, "message", "titre", JOptionPane.QUESTION_MESSAGE);</pre>	
<pre>JOptionPane.showMessageDialog(<i>fenetre</i>, "message", "titre", JOptionPane.PLAIN_MESSAGE);</pre>	

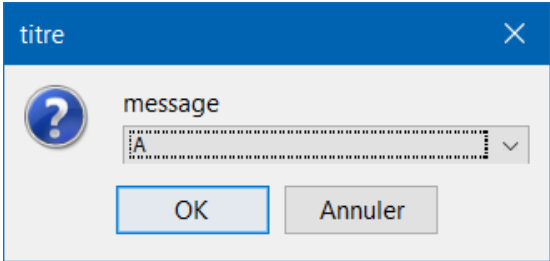

VI.8.2. ConfirmDialog (pour demander une réponse de l'utilisateur parmi des possibilités)

Instruction	Résultat
<pre>int reponse = JOptionPane.showConfirmDialog(<i>fenetre</i>, "message", "titre", JOptionPane.YES_NO_CANCEL_OPTION);</pre>	
<pre>int reponse = JOptionPane.showConfirmDialog(<i>fenetre</i>, "message", "titre", JOptionPane.YES_NO_OPTION);</pre>	
<pre>int reponse = JOptionPane.showConfirmDialog(<i>fenetre</i>, "message", "titre", JOptionPane.OK_CANCEL_OPTION);</pre>	
<pre>int reponse = JOptionPane.showConfirmDialog(<i>fenetre</i>, "message", "titre", JOptionPane.DEFAULT_OPTION);</pre>	

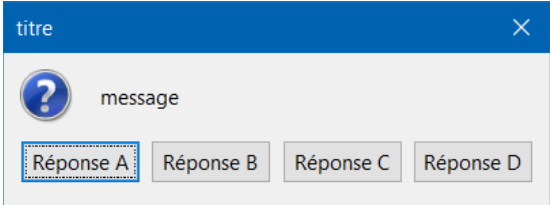
OK renvoie JOptionPane.OK_OPTION qui vaut 0.
 Oui renvoie JOptionPane.YES_OPTION qui vaut 0.
 Non renvoie JOptionPane.NO_OPTION qui vaut 1.
 Annuler renvoie JOptionPane.CANCEL_OPTION qui vaut 2.
 Fermer la fenêtre renvoie JOptionPane.CLOSED_OPTION qui vaut -1.

VI.8.3. InputDialog (pour demander une saisie de l'utilisateur)

Instruction	Résultat
<pre>String reponse = JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.INFORMATION_MESSAGE);</pre>	
<pre>String reponse = JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.WARNING_MESSAGE);</pre>	
<pre>String reponse = JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.ERROR_MESSAGE);</pre>	
<pre>String reponse = JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.QUESTION_MESSAGE);</pre>	
<pre>String reponse = JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.PLAIN_MESSAGE);</pre>	

<pre>String[] choix = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S"}; String reponse = (String) JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);</pre>	
<pre>String[] choix = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T"}; String reponse = (String) JOptionPane.showInputDialog(fenetre, "message", "titre", JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);</pre>	

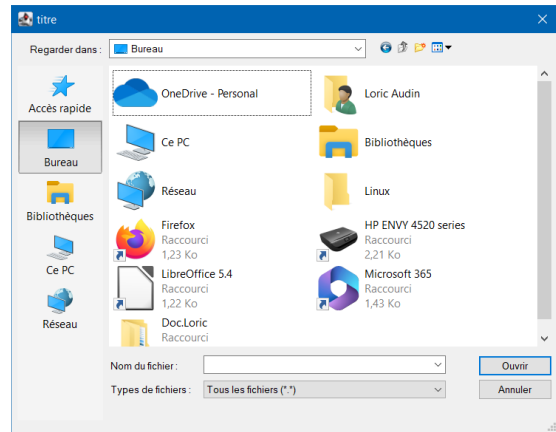
VI.8.4. OptionDialog (pour choisir ses propres options de réponses)

Instruction	Résultat
<pre>String[] choix = {"Réponse A", "Réponse B", "Réponse C", "Réponse D"}; int reponse = JOptionPane.showOptionDialog(fenetre, "message", "titre", JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);</pre>	

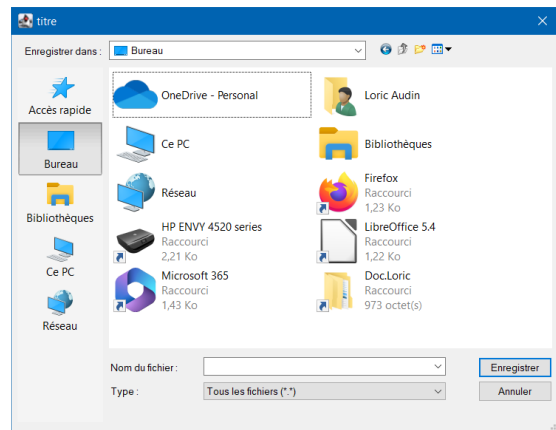
VI.8.5. FileDialog (pour choisir un fichier)

Instruction	Résultat
-------------	----------

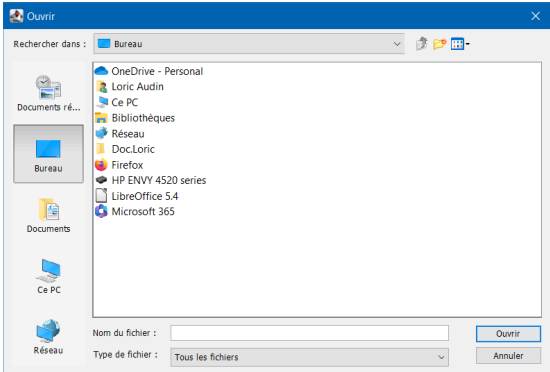
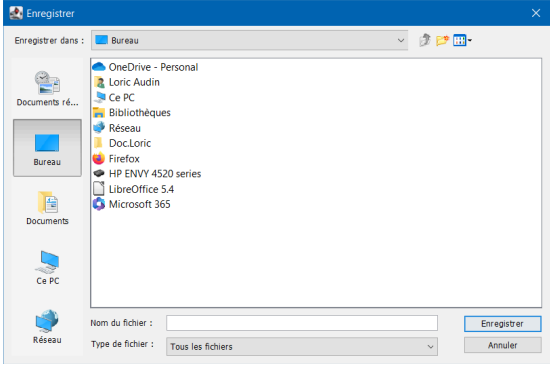
```
FileDialog fd = new
FileDialog(fenetre, "titre",
FileDialog.LOAD);
fd.setVisible(true);
```



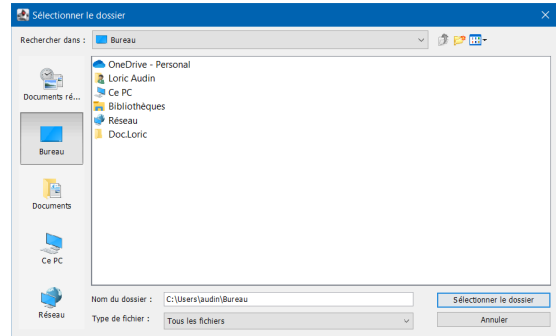
```
FileDialog fd = new
FileDialog(fenetre, "titre",
FileDialog.SAVE);
fd.setVisible(true);
```




VI.8.6. JFileChooser (nouvelle méthode pour choisir un fichier)

Instruction	Résultat
<pre>JFileChooser fc = new JFileChooser(); int reponse = fc.showOpenDialog(fenetre);</pre>	
<pre>JFileChooser fc = new JFileChooser(); int reponse = fc.showSaveDialog(fenetre);</pre>	

```
JFileChooser fc = new JFileChooser();
fc.setFileSelectionMode(
    JFileChooser.DIRECTORIES_ONLY
);
int reponse = fc.showDialog(fenetre,
    "Sélectionner le dossier");
```



VI.8.7. JColorChooser (pour choisir une couleur)

Instruction	Résultat
	

VI.9. Les fenêtres modales