

Mémento

Java

Version 1.0 (créé le 14/01/2024, modifié le 10/12/2024)



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.



Loric Informatique

Table des matières

1. Prise en main.....	6
1.1. Outils nécessaires	6
1.2. Exécuter un programme Java (<i>fichier.java</i>)	6
1.3. Premier code « Hello World! »	6
2. Bases	7
2.1. Syntaxe	7
2.2. Les variables	7
2.2.1. Types de variables	7
2.2.2. Opérations sur les variables	8
2.3. Commentaires	9
2.4. Les tableaux	9
2.5. Conditions	10
2.5.1. Opérateurs de comparaison	10
2.5.2. Tests de conditions	11
2.6. Boucles	11
3. Les fonctions (ou les méthodes de la classe principale)	12
3.1. Créer une fonction retournant une valeur du type « <i>type0</i> »	12
3.2. Retourner une valeur de la fonction	12
3.3. Créer une fonction retournant aucune valeur	12
3.4. Faire un appel à la fonction	13
4. Les classes	13
4.1. Visibilités	13
4.2. Les classes de base	13
4.2.1. Créer une classe	13
4.2.2. Définir la classe dans un objet	14

4.2.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	14
4.2.4. Créer une méthode (dans une classe)	14
4.2.5. Créer une méthode statique (dans une classe)	14
4.2.6. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable dans une classe.....	15
4.2.7. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable en dehors d'une classe.....	15
4.2.8. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe.....	15
4.3. Les classes internes.....	15
4.4. L'héritage	16
4.4.1. Créer une classe héritée d'une autre classe.....	16
4.4.2. Définir la classe dans un objet.....	17
4.4.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	17
4.4.4. Récupérer le nom de la classe héritée.....	17
4.5. Les exceptions.....	17
4.5.1. Exceptions prédéfinies.....	17
4.5.2. Créer une classe d'exception	18
4.5.3. Définir une méthode ou une fonction pouvant renvoyer une exception (en dehors de la classe d'exception).....	18
4.5.4. Provoquer une erreur dans une méthode.....	18
4.5.5. Traiter l'erreur	18
4.6. Les classes abstraites.....	19
4.6.1. Créer une classe abstraite	19
4.6.2. Créer une méthode abstraite	19
4.6.3. Créer une classe héritant d'une classe abstraite	20
4.6.4. Implémenter une méthode abstraite (dans la classe fille).....	20

4.6.5. Définir la classe dans un objet	20
4.7. Les interfaces	21
4.7.1. Créer une interface	21
4.7.2. Créer une méthode abstraite.....	21
4.7.3. Créer une classe implémentant une ou plusieurs interfaces.....	21
4.7.4. Implémenter une méthode abstraite (dans la classe fille)	22
4.7.5. Définir la classe dans un objet.....	22
4.8. Les classes anonymes	23
4.8.1. Manière simple.....	23
4.8.2. Expression lambda (ne fonctionne que si la classe abstraite ne contient qu'une seule méthode abstraite)	23
4.9. Les collections	23
4.9.1. Créer une collection	24
4.9.2. Définir la collection dans un objet.....	24
5. Les énumérations.....	25
5.1. Créer une énumération	25
5.2. Affecter une valeur de l'énumération	25
6. Les instructions.....	26
6.1. Instructions de bases	26
7. Les bibliothèques	27
7.1. Importer une bibliothèque	27
7.2. java.util.....	27
7.2.1. Scanner	27
7.2.2. List.....	27
7.2.3. Set.....	28
7.2.4. Map.....	29
8. Les packages.....	30
9. L'interface graphique.....	31

9.1. Syntaxe	31
9.2. Les différents paramètres de la fenêtre.....	32
9.3. Gestion de la barre de menu	33
9.4. Gestion du panel (zone graphique)	34
9.4.1. Les différents layouts (méthodes de positionnement)	34
9.4.2. Les différents paramètres	35
9.5. Gestion des widgets et des conteneurs	35
9.5.1. Instructions en commun pour tous les widgets et conteneurs	35
9.5.2. Les différents widgets.....	36
9.5.3. Les différents conteneurs	40
9.6. Gestion du menu contextuel apparaissant au clic droit.....	41
9.7. Les événements	41
9.7.1. Les différents écouteurs d'événements.....	42
9.7.2. La gestion des événements	42
9.7.3. Informations sur les événements.....	50
9.8. Les boîtes de dialogues.....	50
9.9. Les fenêtres modales.....	55
9.9.1. Syntaxe	56
9.9.2. Les différents paramètres de la fenêtre modale.....	56

1. Prise en main

1.1. Outils nécessaires

- Un logiciel de codage (Visual Studio Code ou Notepad++)
- Java JDK (version 17 ou supérieure)
- Java Maven (facultatif) (version 3.9.6 ou supérieure)
- Ou un logiciel tout en un (NetBeans (fortement recommandé))

1.2. Exécuter un programme Java (*fichier.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.

1.3. Premier code « Hello World! »

```
public class NomDuProgramme {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

2. Bases

2.1. Syntaxe

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

2.2. Les variables

2.2.1. Types de variables

2.2.1.1. 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	$-3.4 \times 10^{38}f$	$3.4 \times 10^{38}f$	Nombre décimal
double (8o)	Double	-1.7×10^{308}	1.7×10^{308}	Nombre décimal

2.2.1.2. Caractères

Type	Valeurs possibles	Description
char (lo)	1 caractère	Caractère entre '' (\n : retour à la ligne ; \t : tabulation ; \b : retour en arrière ; \f : nouvelle page)
string	Infinité de caractères	Chaîne de caractères entre " "

2.2.1.3. Autres types

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

2.2.2. Opérations sur les variables

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>type variable;</i>	Déclare une nouvelle variable
<i>type variable = valeur;</i>	Déclare une nouvelle variable avec une valeur affectée
int a;	Déclare la variable a comme int
a = 5;	Affecte 5 à une variable
a = a + 3;	Ajoute 3 à une variable
a += 3;	Ajoute 3 à une variable
a++;	Ajoute 1 à une variable

<code>b = a++;</code>	Équivalent à : <code>b = a; a = a + 1;</code>
<code>b = ++a;</code>	Équivalent à : <code>a = a + 1; b = a;</code>
<code>int a = (int) b;</code>	Convertit le nombre décimal <code>b</code> en nombre entier <code>a</code> (cast)
<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 = "chaine";</code>	Déclare une variable contenant une chaîne de caractère (les chaînes peuvent s'additionner avec l'opérateur +)

2.3. Commentaires

```
//Commentaire tenant sur une ligne

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

2.4. Les tableaux

Instruction	Description
<code>type tableau[] = new type[i];</code>	Crée un tableau vide de <code>i</code> éléments
<code>tableau[i]</code>	Renvoie la valeur du tableau à la position <code>i</code> (l'indice de la première valeur est 0), et permet aussi l'écriture d'une autre valeur
<code>int tableau[] = {1, 2, 3};</code> <code>int tableau[] = new int{1, 2, 3};</code>	Crée un tableau de 3 entiers avec des valeurs personnalisées
<code>tableau.length</code>	Renvoie la longueur du tableau

Remarque : En java, une chaîne de caractères n'est pas un tableau.

2.5. Conditions

Une condition renvoie true si elle est respectée et false sinon

2.5.1. Opérateurs de comparaison

Condition	Description de ce que vérifie la condition
$a == b$	a égal à b (seulement en contenu : $1="1"$)
$a === b$	a égal à b (en type et en contenu : $1\neq"1"$)
$a < b$	a strictement inférieur à b
$a > b$	a strictement supérieur à b
$a <= b$	a supérieur ou égal à b
$a != b$	a n'est pas égal à b (seulement en contenu : $1="1"$)
$a !== b$	a n'est pas égal à b (en type et en contenu : $1\neq"1"$)
$a == \text{null}$	Tester si une variable est nulle
$ $	À mettre entre deux conditions, permet d'avoir une des deux conditions qui doit être vraie
$\&\&$	À mettre entre deux conditions, permet d'avoir deux conditions qui doivent être vraies
$!condition$	Ne doit pas respecter la condition

2.5.2. 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>
<pre>a = (condition1) ? 1 : 0;</pre>	Si <i>condition1</i> est vraie, <i>a</i> prend la valeur 1, sinon 0.

2.6. Boucles

Instruction	Description
<pre>for (int i = d; i < f; i++) { instruction1; }</pre>	On répète <i>f-d</i> fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris
<pre>for (int i = d; i < f; i+=p) { instruction1; }</pre>	On répète $(f-d)/p$ fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris avec pour pas égal à <i>p</i>
<pre>for (type elt : tableau) { instruction1; }</pre>	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 la condition soit fausse (peut ne pas être répété)
<code>do { instruction1; } while(condition);</code>	On répète jusqu'à ce que la condition soit fausse (est forcément répété une fois)
<code>break;</code>	Permet de sortir d'une boucle sans la terminer (à éviter si possible)

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

Les fonctions doivent être écrites avant ou après la méthode `main`.

3.1. Créer une fonction retournant une valeur du type « `type0` »

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

3.2. Retourner une valeur de la fonction

```
return variable;
```

3.3. Créer une fonction retournant aucune valeur

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

3.4. 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.

4. Les classes

4.1. 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

4.2. Les classes de base

4.2.1. Créer une classe

```
public class MaClasse {  
  
    private type1 variable1;  
    private type2 variable2;  
  
    public static type3 variableStatique3 = valeur3;  
}
```

```
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.

4.2.2. Définir la classe dans un objet

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

4.2.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

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

4.2.4. Créer une méthode (dans une classe)

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

4.2.5. Créer une méthode statique (dans une classe)

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

4.2.6. 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;
```

4.2.7. 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

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

```
MaClasse.maMethodeStatique(valeur1, valeur2...)
```

```
MaClasse.variableStatique3 = valeur3;
```

4.3. 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.

4.4. 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.

4.4.1. Créer une classe héritée d'une autre classe

```
public class MaClasseHeritee extends MaClasse {  
  
    private type4 variable4;  
    private type5 variable5;  
  
    public static type6 variableStatique6 = valeur6;  
  
    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.

4.4.2. Définir la classe dans un objet

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

ou

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

4.4.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

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

4.4.4. Récupérer le nom de la classe héritée

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

4.5. 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.

4.5.1. 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

4.5.2. Créer une classe d'exception

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

4.5.3. 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;  
}
```

4.5.4. Provoquer une erreur dans une méthode

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

4.5.5. 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.

4.6. 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.

4.6.1. 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;  
    }  
}
```

4.6.2. 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.

4.6.3. Créer une classe héritant d'une classe abstraite

```
public class MaClasseHeritee extends MaClasseAbstraite {
    private type4 variable4;
    private type5 variable5;

    public static type6 variableStatique6 = valeur6;

    visibilite MaClasseHeritee(type1 mVariable1, type2 mVariable2...,
type4 mVariable4, type5 mVariable5...) {
        super(mVariable1, mVariable2...);
        this.variable4 = mVariable4;
        this.variable5 = mVariable5;
    }
}
```

4.6.4. 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.

4.6.5. Définir la classe dans un objet

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

ou

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

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

4.7. 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.

4.7.1. Créer une interface

```
public interface MonInterface {  
    ...  
}
```

4.7.2. 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.

4.7.3. 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 = valeur6;
```

```

        visibilite MaClasseHeritee(type1 mVariable1, type2 mVariable2...,
type4 mVariable4, type5 mVariable5...) {
            super(mVariable1, mVariable2...);
            this.variable4 = mVariable4;
            this.variable5 = mVariable5;
        }
    }
}

```

4.7.4. 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.

4.7.5. Définir la classe dans un objet

```

MonInterface monObjet = new MaClasseAbstraite(valeur1, valeur2...,
valeur4, valeur5...);

```

ou

```

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

```

ou

```

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

```

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.

4.8. 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.

4.8.1. Manière simple

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

4.8.2. Expression lambda (ne fonctionne que si la classe abstraite ne contient qu'une seule méthode abstraite)

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

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

4.9. Les collections

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

4.9.1. Créer une collection

```
public class MaCollection<T1, T2...> {  
  
    visibilite type1 variable1;  
    visibilite type2 variable2;  
    visibilite T1 variable3;  
    visibilite 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.

4.9.2. Définir la collection dans un objet

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


5. Les énumérations

5.1. Créer une énumération

Exemple :

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

5.2. Affecter une valeur de l'énumération

Exemple :

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

6. Les instructions

6.1. 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
<code>System.out.println(variable);</code> <code>System.out.println("Valeur : " + variable);</code>	Affiche une variable dans la console
<code>variable.getClass().getName();</code>	Renvoie le nom de la classe de la variable
<code>chaine1.equals(chaine2);</code>	Vérifie que la première chaîne est égale à la deuxième (fonctionne également avec les objets)
<code>chaine1.concat(chaine2);</code>	Concatène une chaîne avec une autre
<code>chaine.toUpperCase();</code>	Renvoie la chaîne de caractères en majuscule
<code>chaine.toLowerCase();</code>	Renvoie la chaîne de caractères en minuscule
<code>chaine.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>	Récupérer le caractère à la position <i>i</i>

7. Les bibliothèques

7.1. Importer une bibliothèque

```
import bibliotheque.*;
```

Avec *, toutes les fonctions et classes sont récupérée de la bibliothèque. Il est également possible de spécifier à la place le nom de la fonction ou de la classe à importer.

7.2. java.util

Importation : `import java.util.*;`

7.2.1. 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);
```

Importation : `import java.util.Scanner;`

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

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

Importation :

```
import java.util.List;  
import java.util.XXXList;
```

Instruction	Description
<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

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

Importation :

```
import java.util.Set;
import java.util.XXXSet;
```

Instruction	Description
<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
<pre>Iterator itérateur = liste.iterator(); while (itérateur.hasNext()) { typeObjet element = (typeObjet) itérateur.next(); }</pre>	Permet de parcourir la collection

7.2.4. Map

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

Les différentes listes possibles :

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

Importation :

```
import java.util.Map;
import java.util.XXXMap;
```

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

<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

8. Les packages

Les packages permettent de réorganiser les différents fichiers Java dans différents dossiers.

8.1. Définir le package d'un fichier Java

```
package monPackage;
```

ou

```
package monPackage.monSousPackage...;
```

8.2. Importer tous les éléments d'un package

```
import monPackage.*;
```

ou

```
import monPackage.monSousPackage....*;
```

9. L'interface graphique

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

L'interface graphique est réalisée avec les bibliothèques awt et swing.

Importations :

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

9.1. 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;
    }
}

```

9.2. Les différents paramètres de la fenêtre

Instruction	Description
<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 la fenêtre (en haut à gauche et dans la barre des tâches)
<code>this.setSize(<i>Longueur</i>, <i>hauteur</i>);</code>	Redimensionner la fenêtre
<code>this.setResizable(false);</code>	Interdire le redimensionnement de la fenêtre
<code>this.setExtendedState(JFrame.MAXIMIZED_BOTH);</code>	Mettre la fenêtre en plein écran
<code>this.setLocationRelativeTo(null);</code>	Placer la fenêtre au centre de l'écran
<code>this.setLocation(<i>x</i>, <i>y</i>);</code>	Changer la position de la fenêtre
<code>this.setDefaultCloseOperation(<i>actionFermeture</i>);</code>	Changer l'action au clic sur la croix (valeurs possibles : <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 parente, <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
	Renvoie la position du bord haut de la fenêtre (sans prendre en compte

	le bandeau avec le titre et les boutons)
<code>this.getLocation();</code>	Renvoie la position du bord gauche de la fenêtre
<code>this.setVisible(true);</code>	Afficher la fenêtre (ouvre la fenêtre au moment où elle est appelée)

Conseil : Si le style des éléments de Java ne vous plait pas, il est possible de remettre le style du système d'exploitation. Pour cela, entrez les lignes suivantes :

```
UIManager.setLookAndFeel(
    UIManager.getSystemLookAndFeelClassName()
);
```

9.3. Gestion de la barre de menu

Instruction	Description
<code>JMenu menu1 = new JMenu("mon menu");</code> <code>maBarreDeMenu.add(menu1);</code>	Créer un menu
<code>JMenuItem option1 = new JMenuItem("mon option");</code> <code>menu1.add(option1);</code>	Créer un item d'un menu
<code>JMenuItem option1 = new JMenuItem("mon option", new ImageIcon("src/icone.png"));</code> <code>menu1.add(option1);</code>	Créer un item d'un menu avec une icône
<code>JCheckBoxMenuItem option2 = new JCheckBoxMenuItem("mon option");</code> <code>menu1.add(option2);</code>	Créer un item à cocher d'un menu
<code>JRadioButtonMenuItem option3 = new JRadioButtonMenuItem("mon option");</code> <code>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 item puisse effectuer une action, il faut leur ajouter un écouteur. Voir les événements.

9.4. Gestion du panel (zone graphique)

9.4.1. Les différents layouts (méthodes de positionnement)

Instruction	Insertion de widgets	Description
<pre>monPanel.setLayout(new FlowLayout()); monPanel.setLayout(new FlowLayout(FlowLayout.LEFT)); monPanel.setLayout(new FlowLayout(FlowLayout.RIGHT)); monPanel.setLayout(new FlowLayout(FlowLayout.CENTER, espacementH, espacementV));</pre>	<pre>monPanel.add(widget);</pre>	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)
<pre>monPanel.setLayout(new GridLayout(nbLignes, nbColonnes)); monPanel.setLayout(new GridLayout(nbLignes, nbColonnes, espacementH, espacementV));</pre>	<pre>monPanel.add(widget, ligne, colonne);</pre>	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)
<pre>monPanel.setLayout(new BorderLayout()); monPanel.setLayout(new BorderLayout(espacementH, espacementV));</pre>	<pre>monPanel.add(widget, BorderLayout.CENTER); monPanel.add(widget, BorderLayout.NORTH); monPanel.add(widget, BorderLayout.SOUTH); monPanel.add(widget, BorderLayout.WEST); monPanel.add(widget, BorderLayout.EAST);</pre>	Découpe l'écran en 5 régions (avec un espacement horizontal et un espacement vertical si les paramètres sont spécifiées)
<pre>monPanel.setLayout(null);</pre>	<pre>widget.setBounds(x, y, largeur, hauteur);</pre>	Place les widgets de manière absolue grâce à des valeurs personnalisées (fortement déconseillé)

9.4.2. Les différents paramètres

Instruction	Description
TODO	

9.5. Gestion des widgets et des conteneurs

9.5.1. Instructions en commun pour tous les widgets et conteneurs

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.setHeigth(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.getHeigth();</code>	Récupérer la taille du widget

9.5.2. Les différents widgets

9.5.2.1. 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, taille));</code>	Changer le style et la taille du texte (Normal : <code>Font.PLAIN</code> ; Italique : <code>Font.ITALIC</code> ; Gras : <code>Font.BOLD</code>)
<code>label0.setText("message");</code>	Changer le texte

<code>String monTexte = label0.getText();</code>	Récupérer le texte
--	--------------------

9.5.2.2. 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.getScaledInstanc e(largeur, hauteur, 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)

9.5.2.3. 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(booleen);</code>	Débloquer ou bloquer le bouton
<code>bouton0.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>bouton0.setText("message");</code>	Changer le texte
<code>String monTexte = bouton0.getText();</code>	Récupérer le texte

9.5.2.4. JToggleButton (Ajouter un bouton ON/OFF)

Instruction	Description
<code>bouton1 = new JToggleButton("Mon interrupteur", <i>booléen</i>);</code>	Créer le widget (doit être initialisé dans la classe)
<code>bouton1.setEnabled(<i>booléen</i>);</code>	Débloquer ou bloquer le bouton
<code>bouton1.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>bouton1.setSelected(<i>booléen</i>);</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

9.5.2.5. 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, <i>taille</i>));</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(<i>booléen</i>);</code>	Sélectionner ou désélectionner la case

9.5.2.6. 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, taille));</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(booleen);</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

9.5.2.7. JComboBox (Ajouter une liste d'options)

Instruction	Description

9.5.2.8. JList (Ajouter une liste de données)

Instruction	Description

9.5.2.9. JTextField (Ajouter une zone de saisie simple)

Instruction	Description

9.5.2.10. JTextArea (Ajouter une zone de saisie longue)

Instruction	Description

À venir : remplissage des instructions possibles pour les derniers widgets

9.5.3. Les différents conteneurs

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

9.5.3.1. 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 ("titre"));</code>	Ajouter une bordure titrée au panneau

9.5.3.2. JTabbedPane

Instruction	Description

9.5.3.3. JScrollPane

Instruction	Description

9.6. Gestion du menu contextuel apparaissant au clic droit (JPopupMenu)

Instruction	Description
<code>JPopupMenu pop = new JMenu("mon menu"); widget.setComponentPopupMenu(pop);</code>	Créer un menu contextuel
<code>JMenuItem option1 = new JMenuItem("mon option"); pop.add(option1);</code>	Créer un item d'un menu contextuel
<code>JMenuItem option2 = new JMenuItem("mon option", new ImageIcon("src/icone.png")); pop.add(option2);</code>	Créer un item d'un menu contextuel avec une icône
<code>JCheckBoxMenuItem option3 = new JCheckBoxMenuItem("mon option"); pop.add(option3);</code>	Créer un item à cocher d'un menu contextuel
<code>JRadioButtonMenuItem option4 = new JRadioButtonMenuItem("mon option"); pop.add(option4);</code>	Créer un item radio d'un menu contextuel
<code>pop.addSeparator();</code>	Ajouter un trait de séparation

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

9.7. Les événements

9.7.1. Les différents écouteurs d'événements

9.7.1.1. 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.

9.7.1.2. Écouteurs en classe interne

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

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

9.7.1.3. Écouteurs en classe anonyme

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

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

9.7.2.1. 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...

9.7.2.2. 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

9.7.2.3. 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

9.7.2.4. 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

9.7.2.5. 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
componentAdded(ContainerEvent e)	Dès l'ajout d'un composant dans le conteneur
componentRemoved(ContainerEvent e)	Dès la suppression d'un composant dans le conteneur

9.7.2.6. FocusListener

FocusListener détecte des survols de la souris sur des composants.

Composants concernés :

- Component (tous)

Méthode d'enregistrement : addFocusListener()

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

9.7.2.7. 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
itemStateChanged(ItemEvent e)	Dès la sélection ou la désélection d'un élément

9.7.2.8. 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
keyTyped(KeyEvent e)	Dès l'appui sur une touche
keyPressed(KeyEvent e)	Dès le maintien sur une touche
keyReleased(KeyEvent e)	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.

9.7.2.9. DocumentListener

DocumentListener détecte les modifications d'un document

Composants concernés :

- Document

Méthode d'enregistrement : addDocumentListener()

Méthodes à implémenter	Exécution
changedUpdate(DocumentEvent e)	Dès qu'un attribut ou un ensemble d'attributs a été modifié
insertUpdate(DocumentEvent e)	Dès qu'il y a eu une insertion dans le document
removeUpdate(DocumentEvent e)	Dès qu'une partie du document a été supprimée

9.7.2.10. ListSelectionListener

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

Composants concernés :

- JList
- JTable

Méthode d'enregistrement : addListSelectionListener()

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

9.7.2.11. 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.

9.7.2.12. `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 `MouseEventListener`, il est possible d'hériter de `MouseEventAdapter` afin de ne pas être obligé d'implémenter toutes les méthodes.

9.7.2.13. 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.

9.7.2.14. TextListener

TextListener détecte les actions sur une zone de texte.

Composants concernés :

- JTextField

Méthode d'enregistrement : addTextListener()

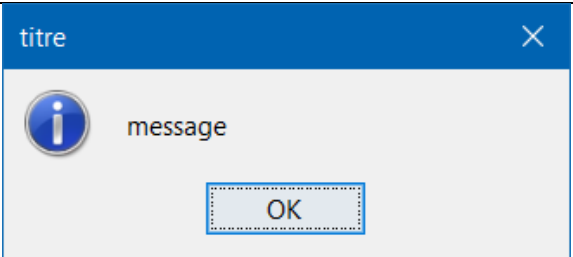
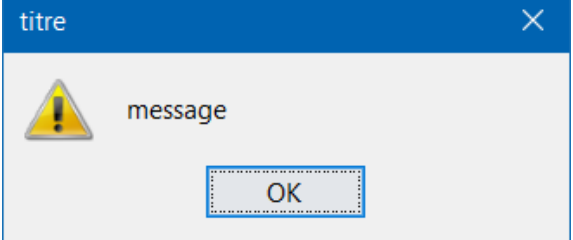
Méthodes à implémenter	Exécution
textValueChanged(TextEvent e)	Dès que la valeur a changé

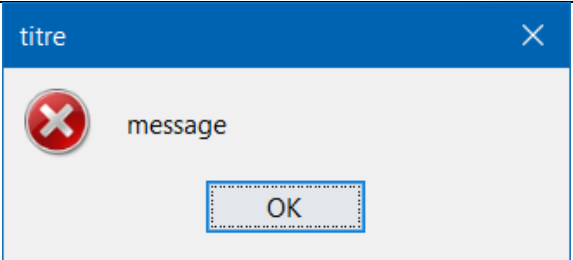
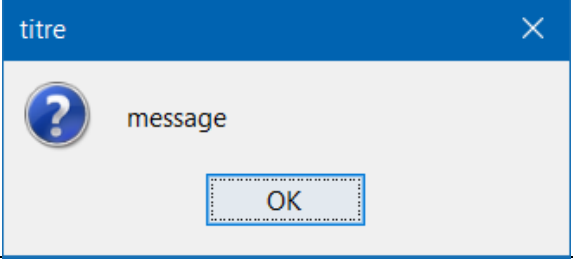
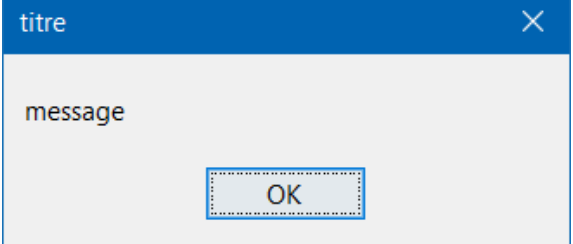
9.7.3. Informations sur les événements

Instructions	Description
e.getSource()	Récupérer le composant source

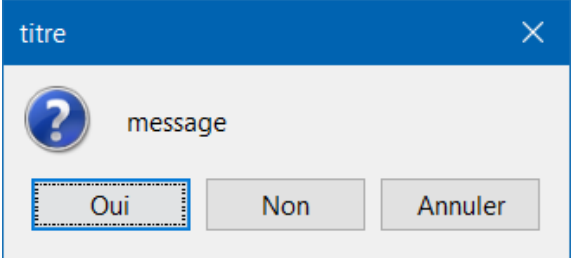
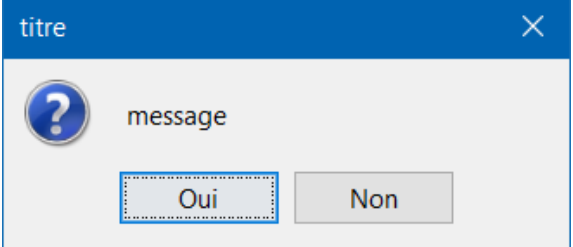
9.8. Les boîtes de dialogues

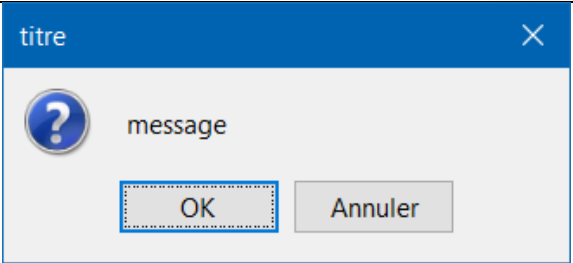
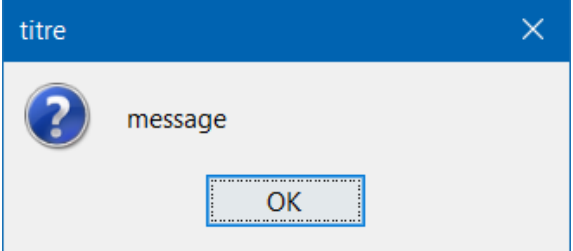
9.8.1. MessageDialog (pour afficher des messages avec un bouton OK)

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

<pre>JOptionPane.showMessageDialog(fe netre, "message", "titre", JOptionPane.ERROR_MESSAGE);</pre>	
<pre>JOptionPane.showMessageDialog(fe netre, "message", "titre", JOptionPane.QUESTION_MESSAGE);</pre>	
<pre>JOptionPane.showMessageDialog(fe netre, "message", "titre", JOptionPane.PLAIN_MESSAGE);</pre>	

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

Instruction	Résultat
<pre>int reponse = JOptionPane.showConfirmDialog(fe netre, "message", "titre", JOptionPane.YES_NO_CANCEL_OPTION);</pre>	
<pre>int reponse = JOptionPane.showConfirmDialog(fe netre, "message", "titre", JOptionPane.YES_NO_OPTION);</pre>	

<pre>int reponse = JOptionPane.showConfirmDialog(fe netre, "message", "titre", JOptionPane.OK_CANCEL_OPTION);</pre>	
<pre>int reponse = JOptionPane.showConfirmDialog(fe netre, "message", "titre", JOptionPane.DEFAULT_OPTION);</pre>	

OK renvoie `JOptionPane.OK_OPTION` qui vaut 0.

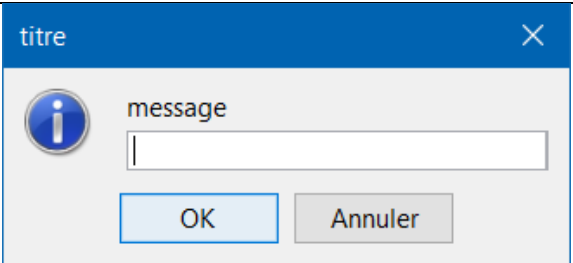
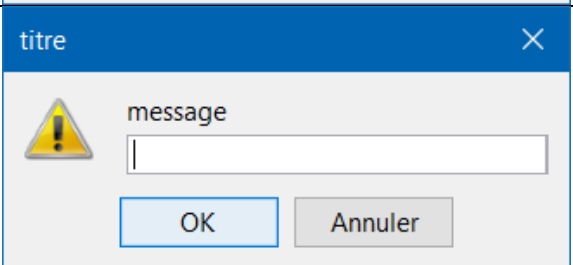
Oui renvoie `JOptionPane.YES_OPTION` qui vaut 0.

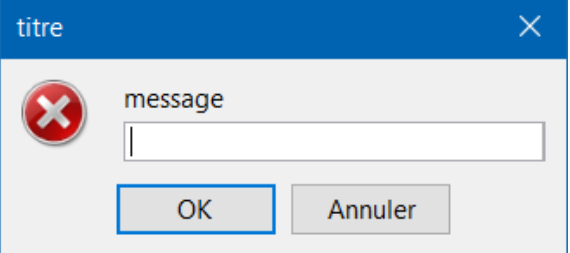
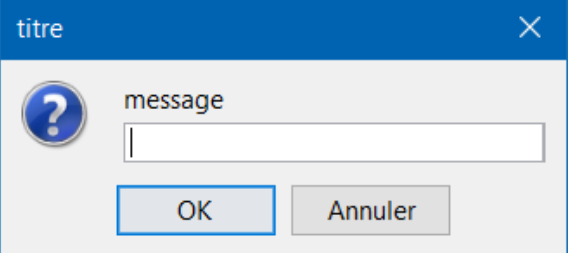
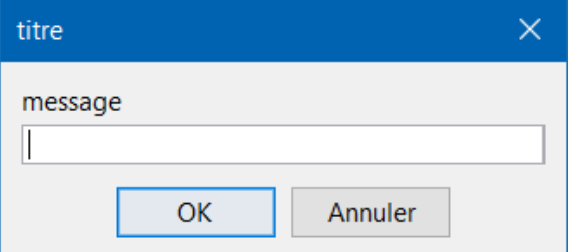
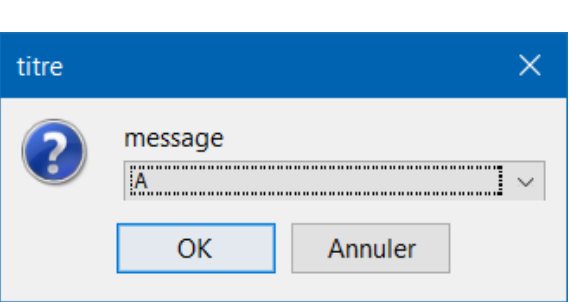
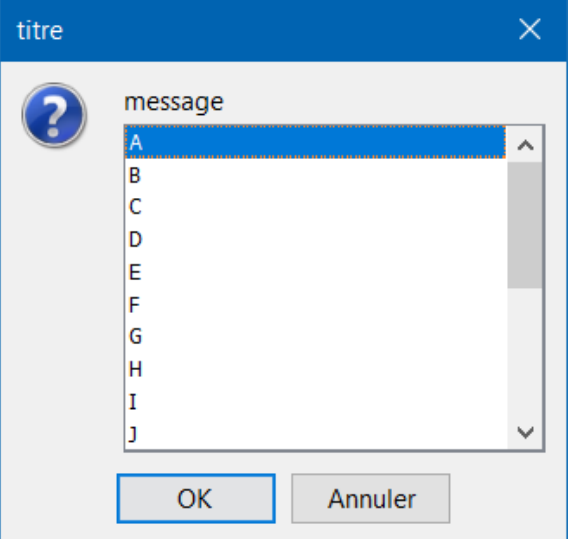
Non renvoie `JOptionPane.NO_OPTION` qui vaut 1.

Annuler renvoie `JOptionPane.CANCEL_OPTION` qui vaut 2.

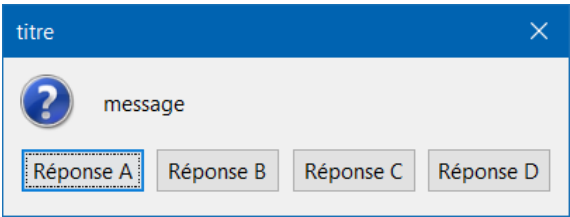
Fermer la fenêtre renvoie `JOptionPane.CLOSED_OPTION` qui vaut -1.

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

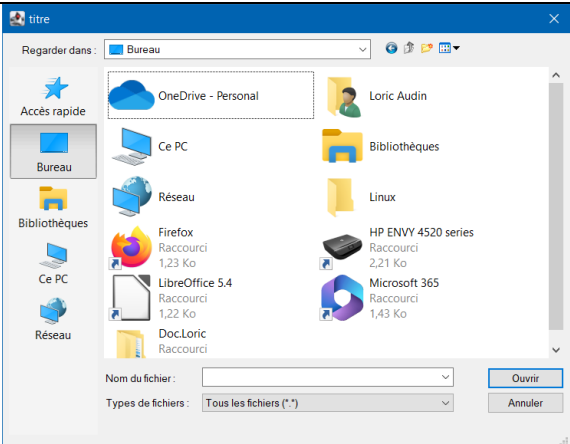
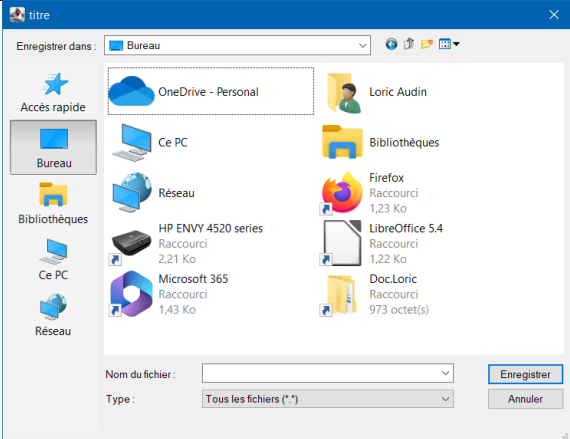
Instruction	Résultat
<pre>String reponse = JOptionPane.showInputDialog(fene tre, "message", "titre", JOptionPane.INFORMATION_MESSAGE) ;</pre>	
<pre>String reponse = JOptionPane.showInputDialog(fene tre, "message", "titre", JOptionPane.WARNING_MESSAGE);</pre>	

<pre>String reponse = JOptionPane.showInputDialog(fene tre, "message", "titre", JOptionPane.ERROR_MESSAGE);</pre>	 <p>A Java Swing dialog box titled "titre" with a close button (X) in the top right corner. It features a red square icon with a white 'X' on the left. The text "message" is displayed above a single-line text input field. At the bottom, there are two buttons: "OK" and "Annuler".</p>
<pre>String reponse = JOptionPane.showInputDialog(fene tre, "message", "titre", JOptionPane.QUESTION_MESSAGE);</pre>	 <p>A Java Swing dialog box titled "titre" with a close button (X) in the top right corner. It features a blue circular icon with a white question mark on the left. The text "message" is displayed above a single-line text input field. At the bottom, there are two buttons: "OK" and "Annuler".</p>
<pre>String reponse = JOptionPane.showInputDialog(fene tre, "message", "titre", JOptionPane.PLAIN_MESSAGE);</pre>	 <p>A Java Swing dialog box titled "titre" with a close button (X) in the top right corner. It has a plain title bar. The text "message" is displayed above a single-line text input field. At the bottom, there are two buttons: "OK" and "Annuler".</p>
<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(fene tre, "message", "titre", JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);</pre>	 <p>A Java Swing dialog box titled "titre" with a close button (X) in the top right corner. It features a blue circular icon with a white question mark on the left. The text "message" is displayed above a list box containing the letters A through S. The letter 'A' is selected. At the bottom, there are two buttons: "OK" and "Annuler".</p>
<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(fene tre, "message", "titre", JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);</pre>	 <p>A Java Swing dialog box titled "titre" with a close button (X) in the top right corner. It features a blue circular icon with a white question mark on the left. The text "message" is displayed above a list box containing the letters A through T. The letter 'A' is selected. At the bottom, there are two buttons: "OK" and "Annuler".</p>

9.8.4. JOptionPane (pour choisir ses propres options de réponses)

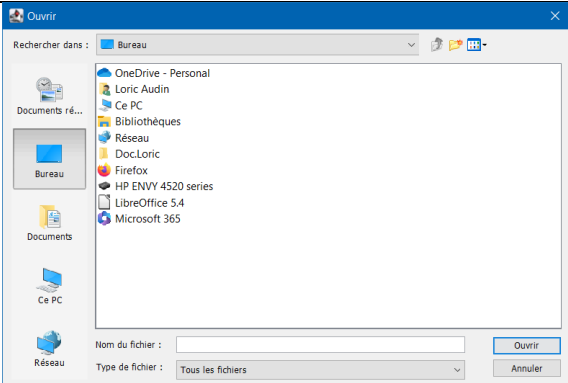
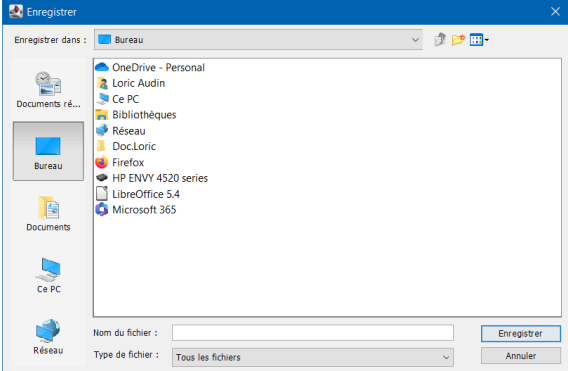
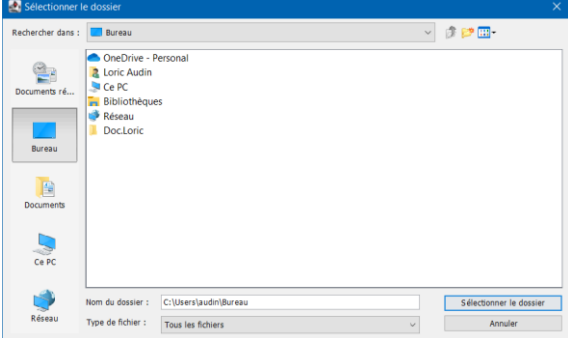
Instruction	Résultat
<pre>String[] choix = {"Réponse A", "Réponse B", "Réponse C", "Réponse D"}; int reponse = JOptionPane.showOptionDialog(fen etre, "message", "titre", JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);</pre>	

9.8.5. FileDialog (pour choisir un fichier)

Instruction	Résultat
<pre>FileDialog fd = new FileDialog(fenetre, "titre", FileDialog.LOAD); fd.setVisible(true);</pre>	
<pre>FileDialog fd = new FileDialog(fenetre, "titre", FileDialog.SAVE); fd.setVisible(true);</pre>	

Récupérer la réponse

9.8.6. 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>	
<pre>JFileChooser fc = new JFileChooser(); fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY); int reponse = fc.showDialog(fenetre, "Sélectionner le dossier");</pre>	

9.8.7. JColorChooser (pour choisir une couleur)

Instruction	Description
TODO	

9.9. Les fenêtres modales

9.9.1. Syntaxe

9.9.2. Les différents paramètres de la fenêtre modale

Instruction	Description
	Interdire le retour sur la fenêtre mère tant que la fenêtre fille n'est pas fermée