Présentation de Java

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Présentation de Java

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:17

Description

Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems présenté officiellement le 23 mai 1995 au *SunWorld*.

Table des matières

1. Aperçu

Le langage Java reprend en grande partie la syntaxe du langage C++. Néanmoins, Java a été 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. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.) qui ont cependant leurs variantes qui héritent de l'objet Object (Integer, Float, ...).

Java permet de développer des applications client-serveur. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, et plus récemment les JSP (JavaServer Pages) qui peuvent se substituer à PHP, ASP.

Java a donné naissance à un système d'exploitation (**JavaOS**), à des environnements de développement (**eclipse**/**JDK**), des machines virtuelles (MSJVM, **JRE**) applicatives multiplate-forme (**JVM**), une déclinaison pour les périphériques mobiles/embarqués (**J2ME**), une bibliothèque de conception d'interface graphique (**AWT**/Swing), des applications lourdes (Jude, Oracle SQL Worksheet, etc.), des technologies web (servlets, applets) et une déclinaison pour l'entreprise (**J2EE**). La portabilité du bytecode Java est assurée par la machine virtuelle Java, et éventuellement par des bibliothèques standard incluses dans un JRE. Cette machine virtuelle peut interpréter le bytecode ou le compiler à la volée en langage machine. La portabilité est dépendante de la qualité de portage des JVM sur chaque OS.

Le nom « Java » n'est pas un acronyme, il a été choisi lors d'un brainstorming. Java veut dire café en argot américain, car c'est la boisson favorite de nombreux programmeurs. Le logo choisi par Sun est d'ailleurs une tasse de café fumant.

2. Open-source et philosophie

Depuis 2006, JAVA est open-source. Sun, puis Oracle, garde toutefois le contrôle de la technologie par le biais d'un catalogue de brevets s'appliquant à Java, ainsi que par le maintien du TCK sous une licence propriétaire.

La société Oracle a acquis en 2009 l'entreprise Sun Microsystems. On peut désormais voir apparaître le logo Oracle dans les documentations de l'api Java.

Ce cours utilisera la dernière version de JAVA SE17, sortie le 14 septembre 2021.

Lors de la création du langage Java, il avait été décidé que ce langage devait répondre à cinq objectifs :

1. simple, orienté objet et familier ;
2. robuste et sûr ;
3. indépendant de la machine employée pour l'exécution ;
4. très performant ;
5. compilé, multi-tâches et dynamique.

Le langage Java s’impose depuis plusieurs années comme la référence dans le monde entier en termes de développement. La philosophie du langage Java repose sur le principe « Write once and run anywhere » (Écrire une fois et exécuter partout).

À de nombreuses reprises Java a connu une perte de popularité dans le domaine de la programmation avec comme cause principale son apprentissage réputé difficile. Pourtant lorsqu’on s’intéresse aux statistiques, le langage reste chaque année parmi les plus utilisés.

En 2020, il y avait près de 8 millions de développeurs JAVA à travers le monde. Dans le rapport RedMonk, il se classe à la 3e position juste après JavaScript et Python.

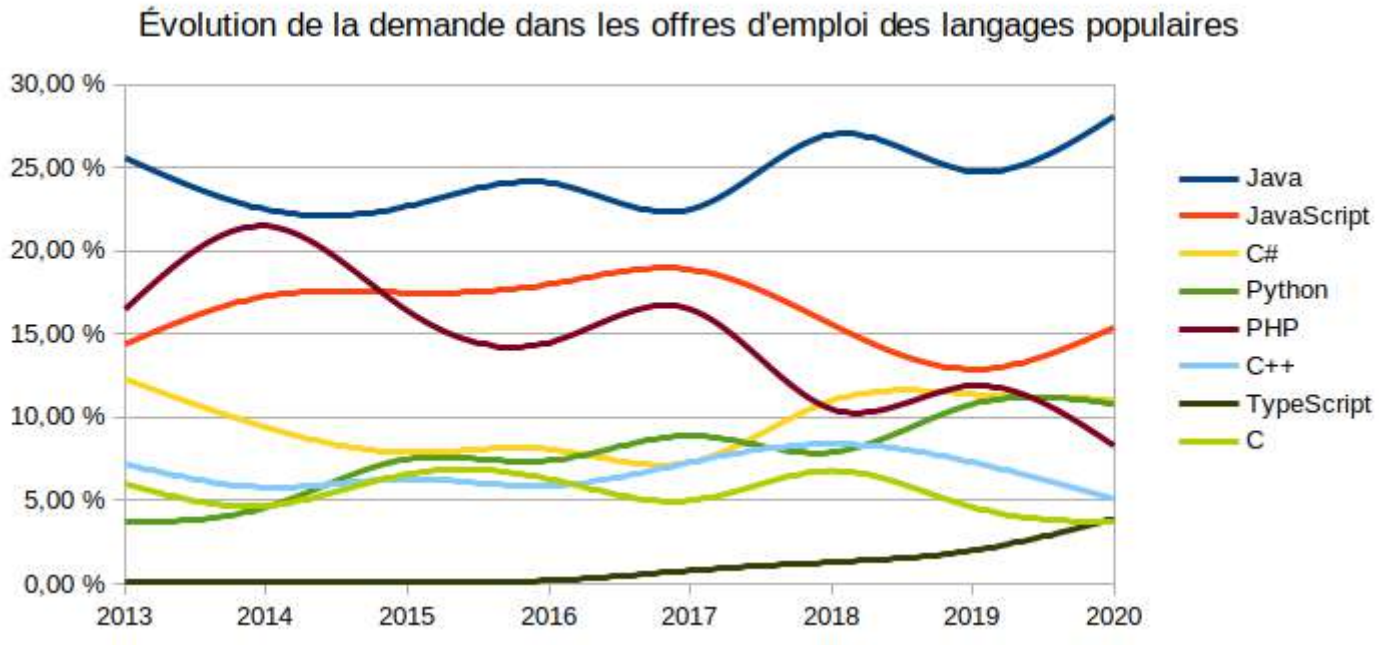
3. Différences Java vs C#

La principale différence entre les deux langages est leur utilisation. Java est principalement dédié au développement d'applications mobiles (plus spécifiquement - Android). Alors que le langage C # se concentre sur le développement Web et le développement de jeux. Ceci dit, il faudrait mentionner que les deux langages ont beaucoup de similarités, ils peuvent tous les deux être utilisés pour le développement Web, mais ils ont également leurs propres utilisations spécifiques!

Une autre chose à noter dans la comparaison C# Java est que, si Java est très flexible et peut être facilement utilisé pour le développement multi-plateformes, C # de son côté manque de paramètres suffisants pour pouvoir le faire il est principalement utilisé par les programmeurs .Net.

Plus de détail sur les différences.

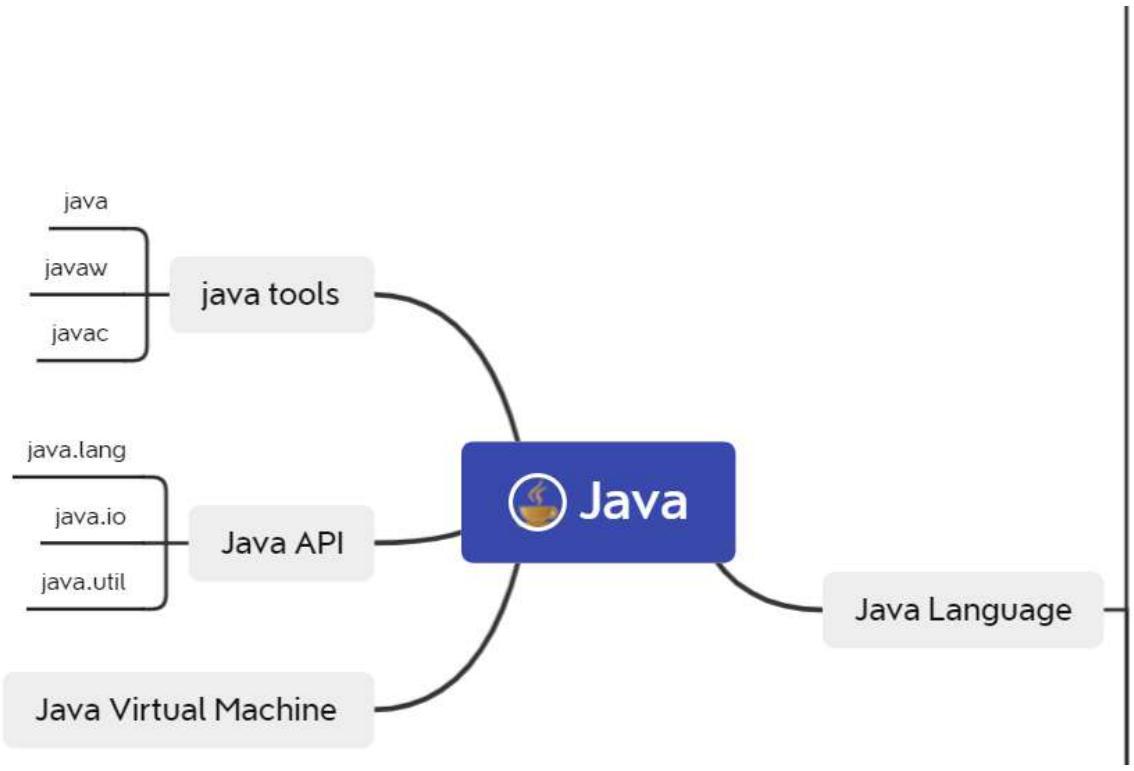
4. Popularité

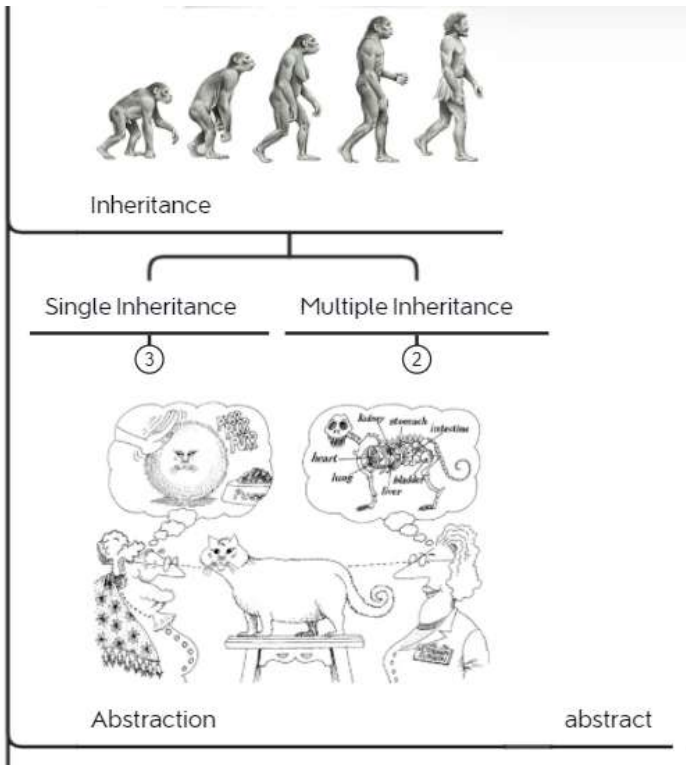


Java est toujours la star de popularité dans les offres d'emploi, maintenant une première place continue sur toutes ces années et creusant même l'écart au fil du temps. Mais les salaires proposés sont corrects, sans plus.

Source : developpez.com

5. Panorama MindMap





Installer Eclipse et JDK

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Installer Eclipse et JDK

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:19

Description

Pour utiliser JAVA vous avez besoin d'un compilateur et d'un IDE.

Compilateur

Nous vous conseillons d'utiliser le compilateur JDK distribué par la société Oracle. Il est gratuit. Utilisez le JDK (JAVA Development Kit) en édition standard (SE). Attention, le runtime (JRE--JAVA Runtime Environment) ne suffit pas.

Vous pouvez le trouver à l'adresse: http://www.oracle.com/technetwork/java/javase/downloads/index.html .

Environnement de développement

Pour le cours, nous vous conseillons d'utiliser l'environnement de développement **eclipse** qui est un environnement puissant.

Vous pouvez télécharger la version **Eclipse IDE for Java EE Developers** sur le site http://eclipse.org/downloads/

Table des matières

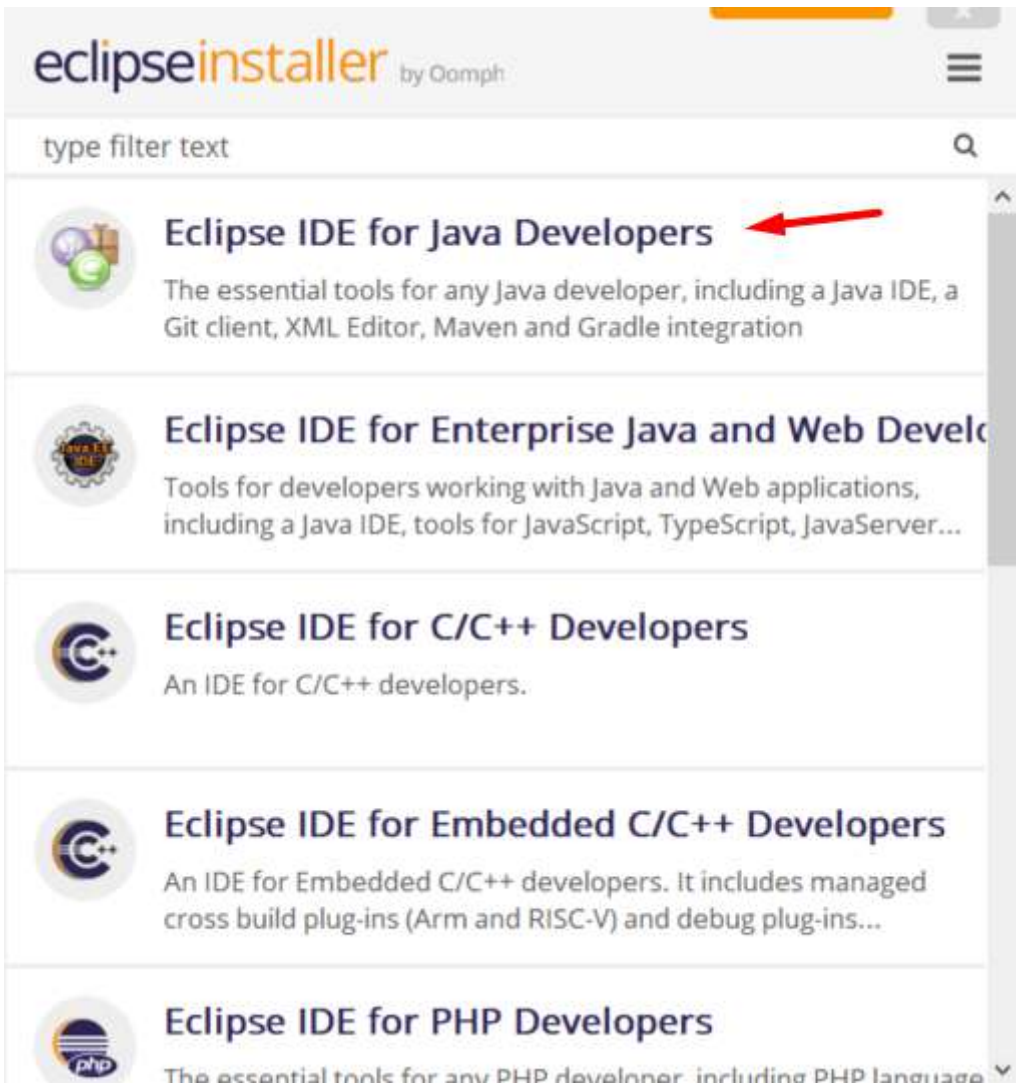
1. L'installation du JDK

Ce document commence au moment où vous avez téléchargé le fichier `jdk-17_windows-x64_bin.msi`. En cliquant sur l'icône de ce fichier, vous lancez l'exécution de la procédure d'installation.

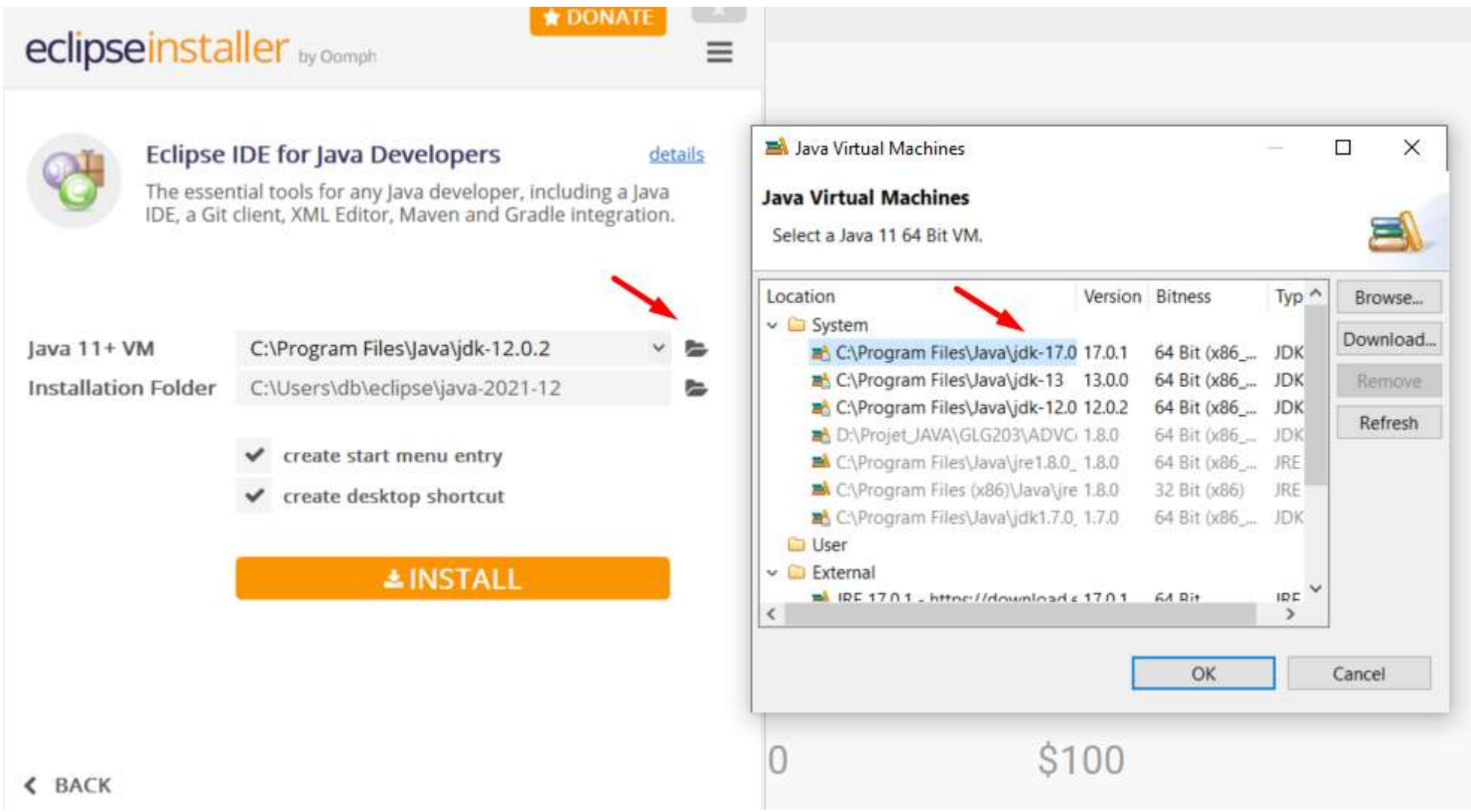
Actionnez le bouton `Next`. Cela provoque une nouvelle installation qui peut prendre elle aussi un certain temps.

Il faut maintenant actionner le bouton `Finish`. Cela fait disparaître la fenêtre. Votre compilateur est installé. Pour aller plus loin vous pouvez consulter la documentation du compilateur. Par exemple il est possible de compiler un fichier txt écrit en JAVA en ligne de commande : https://docs.oracle.com/en/java/javase/17/index.html

2. Installation de Eclipse

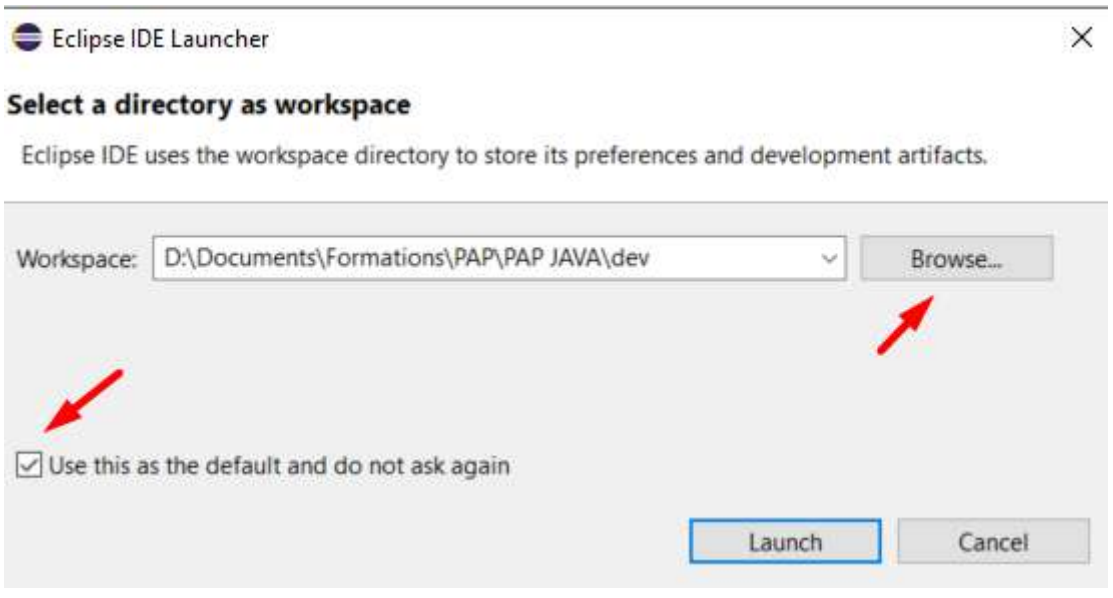


Nous utiliserons la dernière version de JDK :

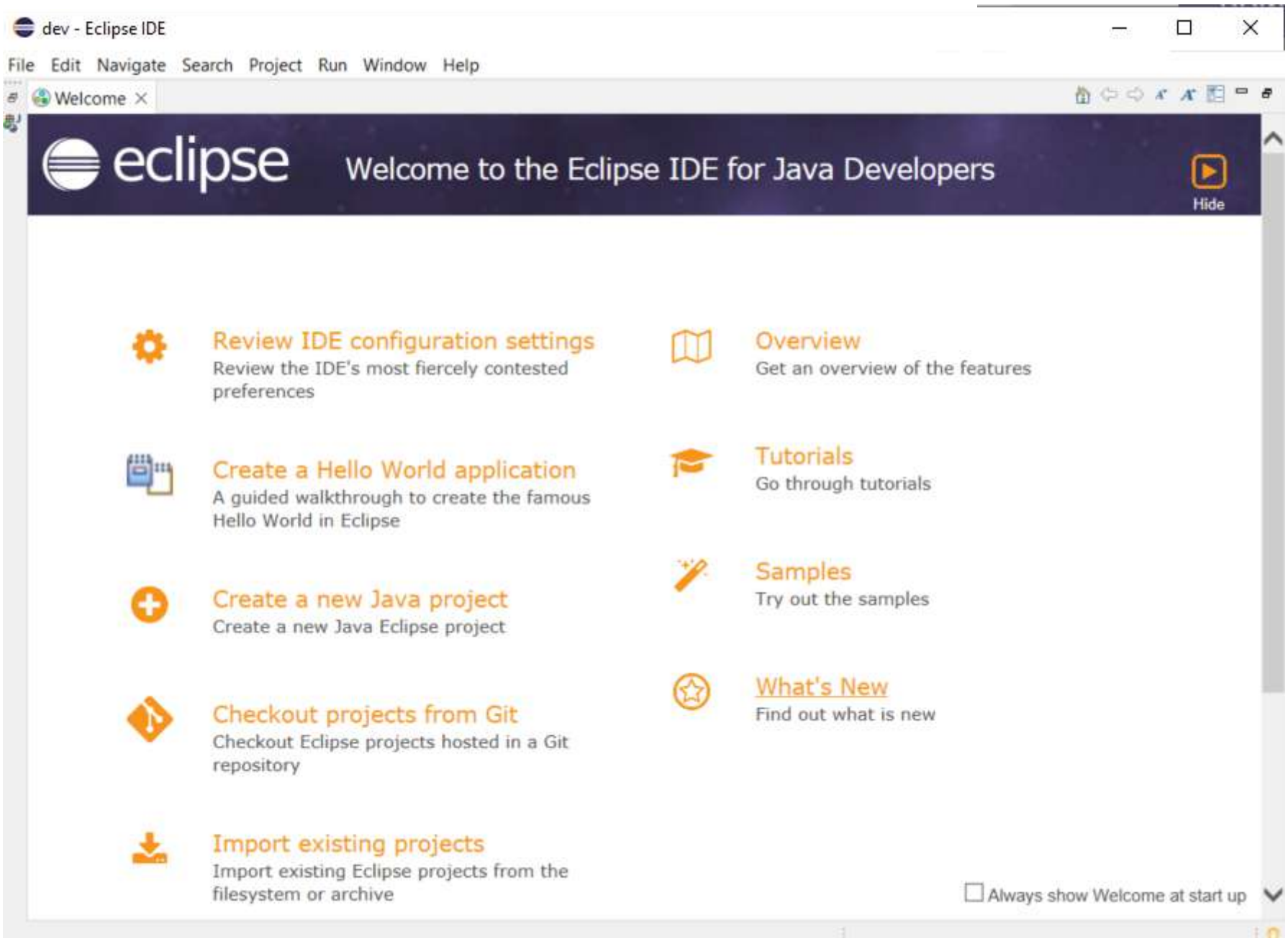


A la fin de l'installation nous allons lancer Eclipse :

Déterminer un espace de travail cohérent :

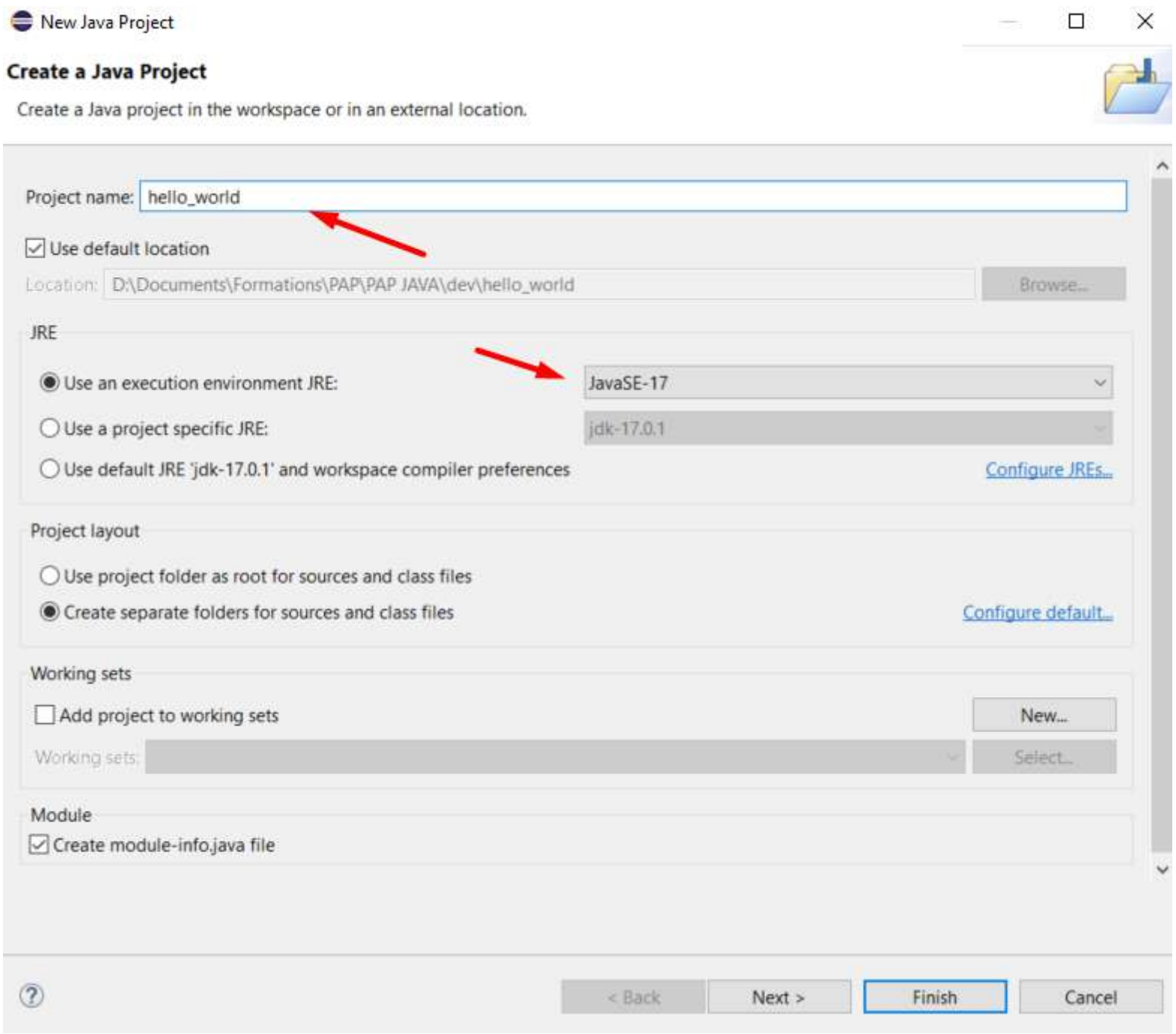


Vous arrivez sur le portail d'Eclipse :


dev - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to the Eclipse IDE for Java Developers
Hide
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Overview
Get an overview of the features
Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse
Tutorials
Go through tutorials
Samples
Try out the samples
Create a new Java project
Create a new Java Eclipse project
What's New
Find out what is new
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Always show Welcome at start up

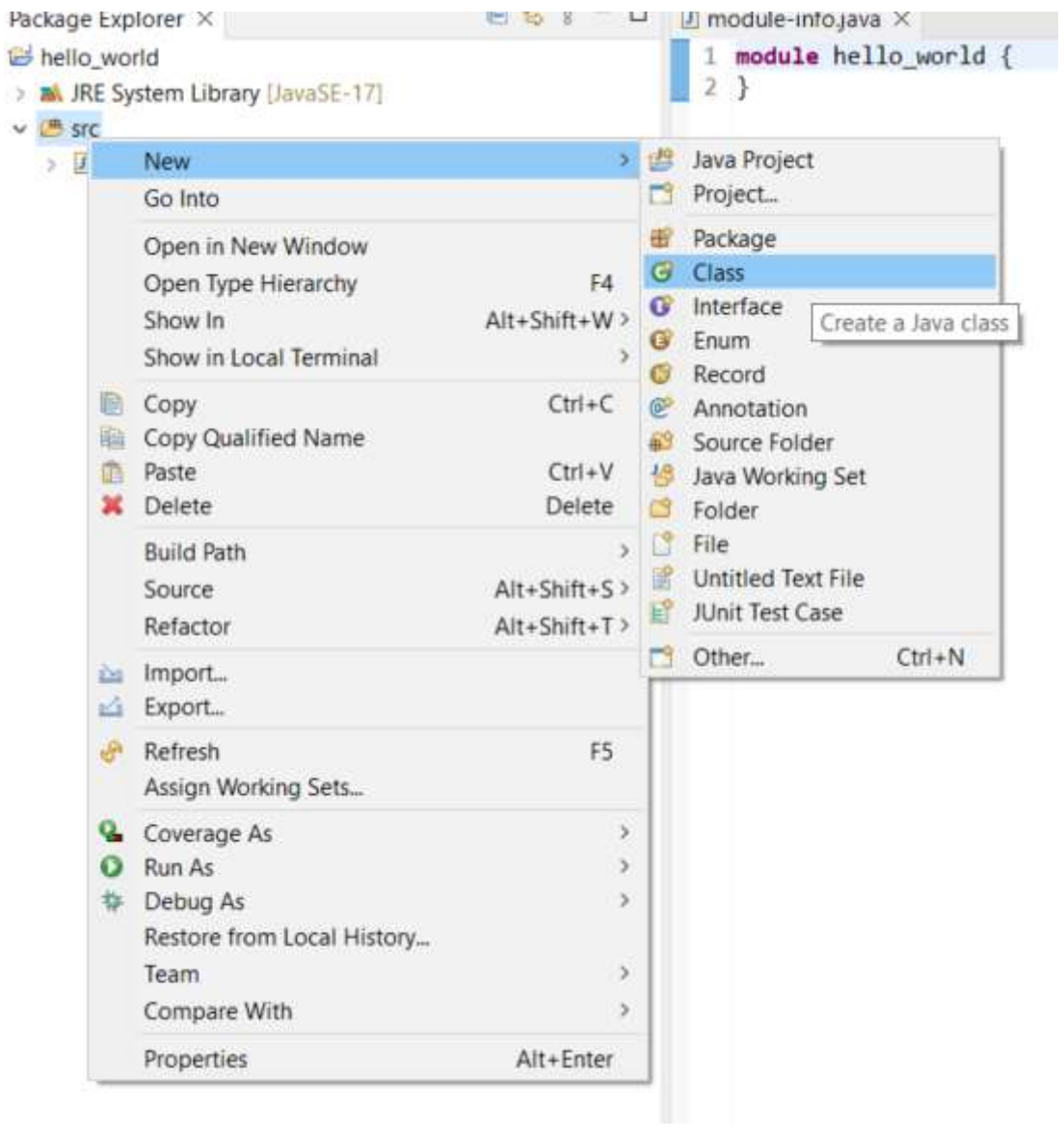
3. Hello World

Nous allons créer notre premier projet JAVA, aller sur Create a Java Project :

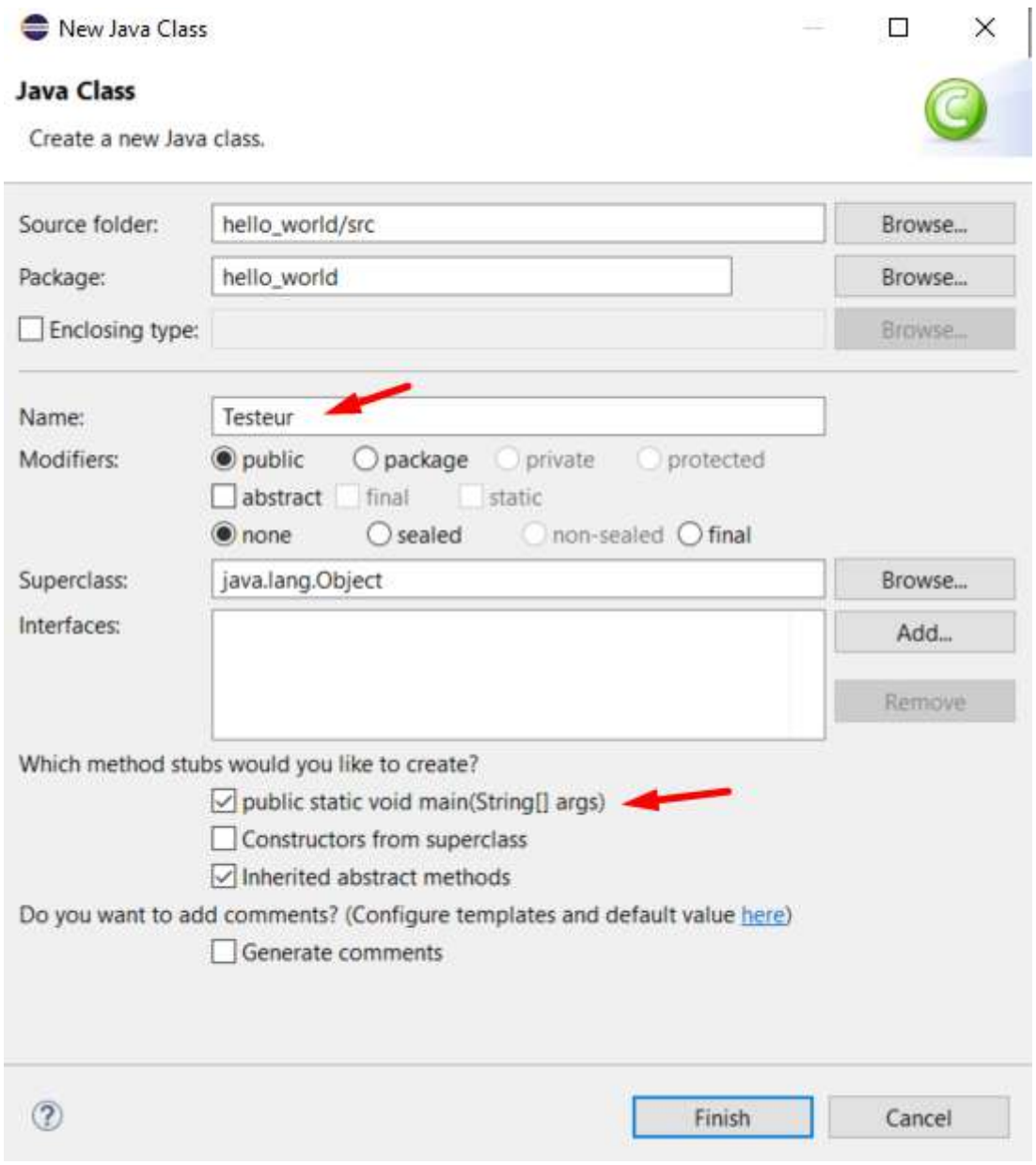


Vérifier le compilateur sélectionné puis Finish.

Nous allons créer notre premiere class, src > new class :



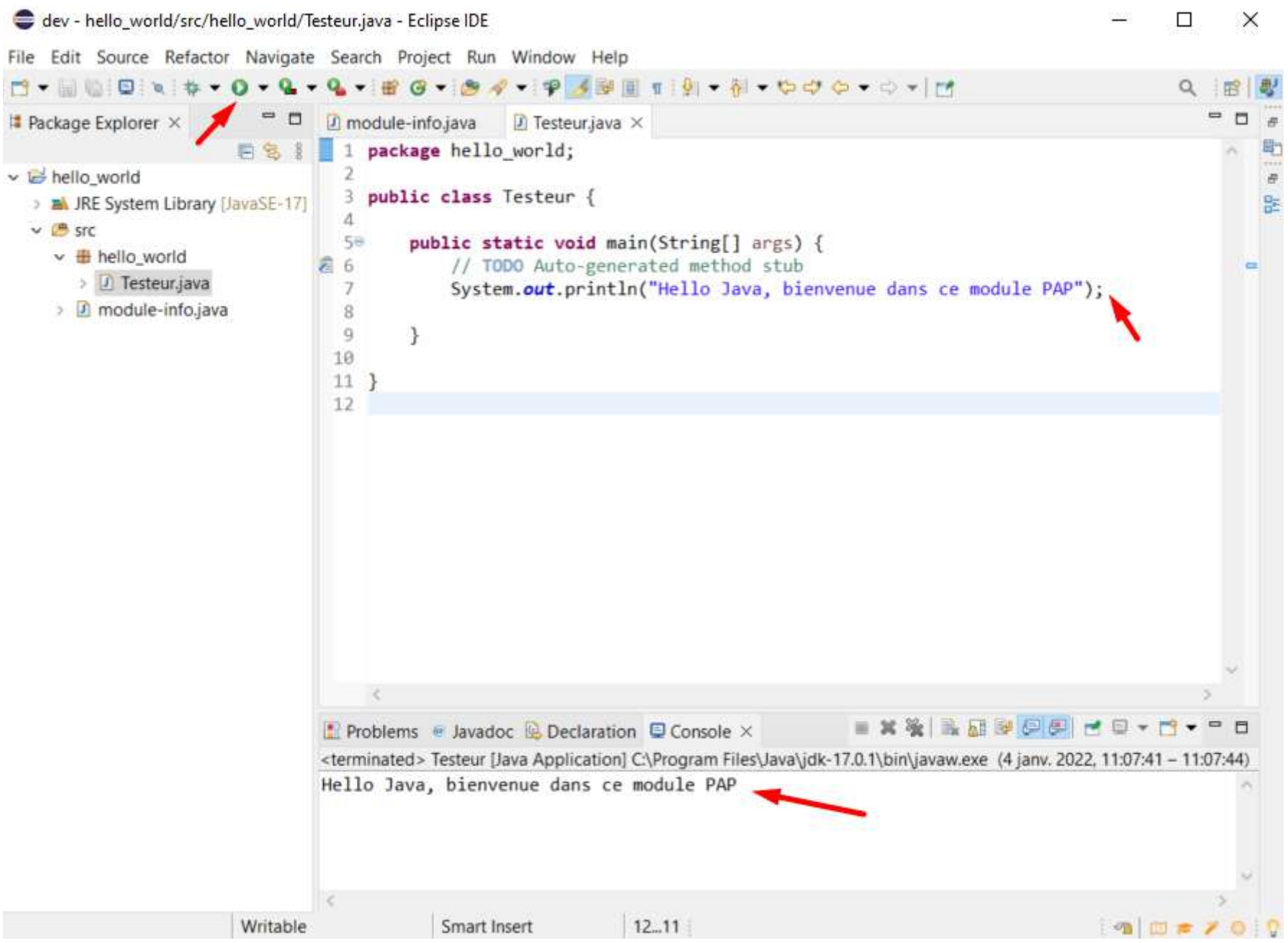
Nous allons l'appeler Testeur, cocher la methode static main :



Ajouter le code suivant :

```java
package hello_world;

public class Testeur {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("Hello Java, bienvenue dans ce module PAP");

	}

}
```

Vérifier l'execution du code dans la console :


dev - hello_world/src/hello_world/Testeur.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
hello_world
JRE System Library [JavaSE-17]
src
hello_world
Testeur.java
module-info.java
module-info.java
Testeur.java
1 package hello_world;
2
3 public class Testeur {
4
5 public static void main(String[] args) {
6 // TODO Auto-generated method stub
7 System.out.println("Hello Java, bienvenue dans ce module PAP");
8
9 }
10
11 }
12
Problems Javadoc Declaration Console
<terminated> Testeur [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (4 janv. 2022, 11:07:41 – 11:07:44)
Hello Java, bienvenue dans ce module PAP
Writable
Smart Insert
12...11

4. Compilation

Un programme java se tape dans un fichier texte ayant l'extension .java. Pour créer ce fichier, il faut un éditeur de texte (par exemple notepad++) ou un environnement de programmation intégré qui comprend un éditeur (par exemple Eclipse).

Ce programme a une forme compréhensible pour un être humain, mais il n'est pas directement exécutable sur machine, il faut le traduire dans un autre format. C'est le compilateur qui assure cette traduction. Dans un terminal, on utilise la commande javac.

Dans un environnement intégrer, on actionne un bouton ou un choix de menu déroulant. La compilation peut échouer s'il y a des erreurs. Il faut alors corriger le programme dans l'éditeur et recommencer la compilation.

S'il n'y a pas d'erreur, le compilateur crée un certain nombre de fichiers avec l'extension .class qui contient le code machine. En java, ce code machine n'est pas directement exécutable par le processeur, il doit être exécuté par un interpréteur de code Java. C'est ce qu'on appelle le runtime java.

Dans un terminal, c'est la commande java. Dans un environnement intégré, c'est un bouton run ou un choix de menu déroulant.

Pour les exemples de ce cours, on utilise une classe appelée Terminal qui réalise les entrées- sorties écran-clavier. Cette classe est à télécharger sur le chapitre 1.1 du cours. Le but de cette classe est notamment de faciliter les lectures au clavier.

Pour compiler les exemples du cours (que vous pouvez copier-coller), vous devez copier le fichier Terminal.java dans le même dossier que le programme.

Premiers pas avec le debugger Eclipse

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Premiers pas avec le debugger Eclipse

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:20

Description

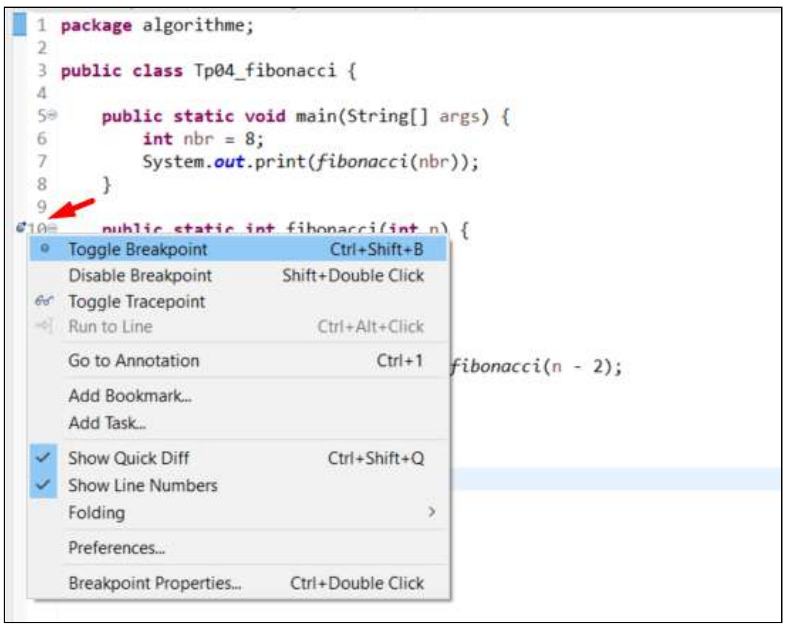
Un debugger permet d’exécuter un programme pas-à-pas afin d’en vérifier le comportement et l’état. Voici la démarche simplifiée à suivre.

Table des matières

1. Ajouter/Supprimer un point d’arrêt

Un point d’arrêt permet de signaler au debugger où se mettre en pause. Un point d’arrêt est signalé par le symbole (rond bleu).

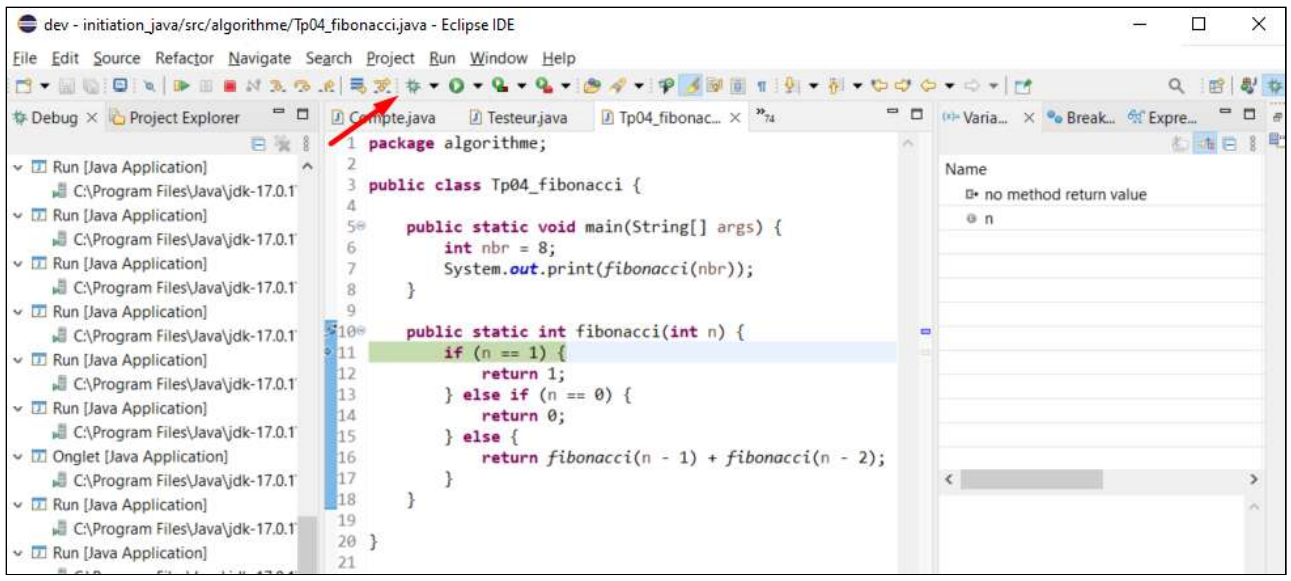
- Ajouter un point d’arrêt : **clic droit sur la marge gauche**, au niveau de la ligne qui nous intéresse, puis choisir **Toggle Breakpoint** ou tout simplement **double cliquer sur la marge gauche** au même niveau
- Supprimer un point d’arrêt : **clic droit sur le point d’arrêt** ( ), puis **Toggle Breakpoint**



2. Lancer/Stopper le debugger

Une fois un point d'arrêt défini, il est temps de lancer le debugger d'Eclipse. Lorsque le debugger est lancé, Eclipse vous propose de passer en perspective *Debug* (*Confirm Perspective Switch*). Il faut accepter, ce qui changera légèrement l'agencement des vues dans Eclipse. Il suffit ensuite d'attendre que l'exécution atteigne le point d'arrêt, pour inspecter l'état des variables, puis éventuellement relancer l'exécution.

- Lancer le debugger : **clic sur l'icône**  (*Debug*) ou clic droit sur la classe contenant la méthode main puis *Debug As* > *Java Application* dans le menu contextuel



3. Inspecter la variable

Une fois le debugger en pause, il est possible d'inspecter l'état d'une variable dans la vue **Variable** (en haut à droite de la fenêtre, par défaut). Par exemple, dans la figure suivante, la variable `n` est égale à 8.

Une fois les variables souhaitées inspectées, vous pouvez soit continuer l'exécution pour vérifier d'autres points d'arrêts, ou bien stopper l'exécution.

- Relancer le debugger : **clic sur l'icône** (*Resume*)
- Arrêter l'exécution : **clic sur l'icône** (*Terminate*)
- Revenir à la perspective *Java* de base : **clic sur l'icône** (*Java Perspective*)

Rappel des notions de base en algorithme

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Rappel des notions de base en algorithme

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:20

Description

Le PAP Initiation JAVA est consacré à l’apprentissage des concepts de la programmation objet tels qu’ils sont mis en oeuvre en Java. Cet apprentissage suppose connus un certain nombre de notions de base étudiées dans le socle commun CDA, comme l'algorithme et la POO. Le présent chapitre est consacré à un bref rappel de ces notions.

Table des matières

1. Variables et types

La structure minimale d'un programme Java consiste en une classe comportant une méthode `main`. Cette méthode contient une suite d'instructions.

```java
public class testeur {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		 System.out.println("Hello Java");
	}
}
```

Une variable associe un nom à un espace mémoire destiné à recevoir une valeur d'un type donné. Cette valeur peut varier au fil de l'exécution du programme et elle peut être inconnue au moment où l'on écrit le programme. On la désigne par le nom de la variable.

Une variable est déclarée avec son type suivi de son nom et éventuellement sa valeur initiale.

```
int nombre; // type int, nom nombre

int ligne = 5; // type int, nom ligne, valeur initiale 5
```

Les types utilisables pour déclarer une variables sont les types de base, les types tableaux (cf. ci-dessous) et les types objets.

Les types de base :

- int : nombres entiers
- double : nombres à virgule (approximation finie des nombres réels)
- char : caractères
- boolean : les valeurs de vérité true et false
- d'autres types entiers et nombre à virgule (non utilisés dans ce cours) : byte, long, float.

Les types objets comportent notamment les chaînes de caractère (type String).

On donne une nouvelle valeur à une variable au moyen d'une instruction d'affectation. La nouvelle valeur remplace ce qui était dans la variable et qui est alors perdu.

```
nombre = 17;

nombre = nombre + 1;
```

Exemples de déclarations, d'affectations et d'entrées-sorties pour les différents types :

```java
public class Testeur {

	public static void main(String[] args) {
		// declarations
		int entier;
		double avirgule;
		char carac;
		boolean b;
		String ch;

		// affectation de valeurs
		entier = 17;
		avirgule = 14.369;
		carac = 'u';
		b = true;
		ch = "Bonjour";


		// affectation d'expressions avec opérateurs
		entier = 4 * entier + 12;
		avirgule = avirgule / 12.36;
		b = b || true; // ou logique
		ch = "salut " + "les copains"; // concatenation de chaines

		// affichage console methode traditionnelle
		System.out.println(entier);
		System.out.println(avirgule - 1.1);
		System.out.println(b);
		System.out.println(ch + " !");

		// saisie au clavier
		Terminal.ecrireStringln("Saisir un entier :");
		entier = Terminal.lireInt();
		Terminal.ecrireStringln("Saisir un nombre à virgule :");
		avirgule = Terminal.lireDouble();
		Terminal.ecrireStringln("Saisir un caractère :");
		carac = Terminal.lireChar();
		Terminal.ecrireStringln("Saisir un boolean :");
		b = Terminal.lireBoolean();
		Terminal.ecrireStringln("Saisir un string :");
		ch = Terminal.lireString();

		// affichage à l'écran méthode class Terminal
		Terminal.ecrireIntln(entier);
		Terminal.ecrireDoubleln(avirgule - 1.1);
		Terminal.ecrireCharln(carac);
		Terminal.ecrireBooleanln(b);
		Terminal.ecrireStringln(ch + " !");

	}

}
```

1.1. Class Terminal

Dans ce cours, nous utilisons une classe spécifique pour réaliser les entrées-sorties écran/clavier, avec des méthodes pour les types int, double, char, boolean et String. Pour lire une valeur au clavier, on utilise les méthodes `Terminal.lireInt`, `Terminal.lireDouble`, etc. Pour écrire une valeur à l'écran, on utilise les méthodes `Terminal.ecrireInt` et `Terminal.ecrireIntln`. La première écrit simplement la valeur et la seconde écrit la valeur puis passe à la ligne. La valeur est donnée en paramètre, entre parenthèses. Par exemple : `Terminal.ecrireInt(147).`

```java
package hello_world;

import java.io.*;

public class Terminal {
	static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

	public static String lireString() // Lire un String
	{
		String tmp = "";
		try {
			tmp = in.readLine();
		} catch (IOException e) {
			exceptionHandler(e);
		}
		return tmp;
	} // fin de lireString()

	public static int lireInt() // Lire un entier
	{
		int x = 0;
		try {
			x = Integer.parseInt(lireString());
		} catch (NumberFormatException e) {
			exceptionHandler(e);
		}
		return x;
	}

	public static boolean lireBoolean() // Lire un entier
	{
		boolean b = true;
		try {
			b = Boolean.valueOf(lireString()).booleanValue();
		} catch (NumberFormatException e) {
			exceptionHandler(e);
		}
		return b;
	}

	public static double lireDouble() // Lire un double
	{
		double x = 0.0;
		try {
			x = Double.valueOf(lireString()).doubleValue();
		} catch (NumberFormatException e) {
			exceptionHandler(e);
		}
		return x;
	}

	public static char lireChar() // Lire un caractere
	{
		String tmp = lireString();
		if (tmp.length() == 0)
			return '\n';
		else {
			return tmp.charAt(0);
		}
	}

	public static void ecrireString(String s) { // Afficher un String
		try {
			System.out.print(s);
		} catch (Exception ex) {
			exceptionHandler(ex);
		}
	}

	public static void ecrireStringln(String s) // Afficher un String
	{
		ecrireString(s);
		sautDeLigne();
	} // fin de ecrireStringln()

	public static void ecrireInt(int i) // Afficher un entier
	{
		ecrireString("" + i);
	}

	public static void ecrireIntln(int i) // Afficher un entier
	{
		ecrireString("" + i);
```

```
		sautDeLigne();
	}

	public static void ecrireBoolean(boolean b) {
		ecrireString("" + b);
	}

	public static void ecrireBooleanln(boolean b) {
		ecrireString("" + b);
		sautDeLigne();
	}

	public static void ecrireDouble(double d) // Afficher un double
	{
		ecrireString("" + d);
	}

	public static void ecrireDoubleln(double d) // Afficher un double
	{
		ecrireDouble(d);
		sautDeLigne();
	}

	public static void ecrireChar(char c) // Afficher un caractere
	{
		ecrireString("" + c);
	}

		public static void ecrireCharln(char c) // Afficher un caractere
		{
			ecrireChar(c);
			sautDeLigne();
		}

		public static void sautDeLigne() {
			try {
				System.out.println();
			} catch (Exception ex) {
				exceptionHandler(ex);
			}
		}

		protected static void exceptionHandler(Exception ex) {
			TerminalException err = new TerminalException(ex);
			throw err;
		}

		public static void ecrireException(Throwable ex) {
			ecrireString(ex.toString());
			ex.printStackTrace(System.err);
		}
}

class TerminalException extends RuntimeException {
		/**
		 *
		 */
		private static final long serialVersionUID = 1L;
		Exception ex;

		TerminalException(Exception e) {
			ex = e;
		}
}
```

1.2. TP

- Créer une `class Types` avec une `methode static testeur`.
- Ajouter dans cette methode le code précédemment utilisé dans main.
- Appeler cette methode directement dans main de la `class Testeur`.

2. Booléens, If et boucles

L'instruction conditionnelle (if) et les boucles (for, while) permettent de faire varier l'exécution du programme en fonction de la valeur d'une condition. Une condition consiste en un calcul (expression) donnant un résultat booléen (true ou false).

Un exemple d'expression booléenne consiste en une combinaison de valeurs booléennes au moyen de connecteurs logiques :

- et, noté && en java
- ou, noté || en java
- non, noté ! en java

Un autre exemple de calcul booléen consiste en une comparaison entre valeurs d'autres types :

- test d'égalité, noté == en java
- test de différence, noté != en java
- comparaisons d'ordre : <, <, <= et >=

Voici quelques expressions booléennes. Ces expressions peuvent apparaître en tant que condition dans un if ou une boucle, mais elle peut également être affectée à une variable de type boolean comme c'est le cas ici.

```
package hello_world;

public class Bools {

	public static void testeur() {

		boolean b1 = true;
		boolean b2;
		Terminal.ecrireString("Entrez la valeur de b2 (true ou false): ");
		b2 = Terminal.lireBoolean();
		int x = 3;
		b1 = b1 && (b1 || b2);
		b1 = x < 10;
		b1 = (x == 10); // test si X est egal a 10
		b2 = (x != 10) && b1;
		b1 = ('a' < 'Z') || (1.2 * 38 == 57.6);
		Terminal.ecrireBooleanln(b1);
		Terminal.ecrireBooleanln(b2);

	}
}
```

2.1. If

L’instruction if a la structure suivante :

```
if (condition){
instruction-1;
instruction-1;
...
instruction-n;
}else{
instruction-n+1;
...
instruction-m;
}
```

Si la condition est vraie (le résultat du calcul vaut `true`), alors les instructions `instruction-1` à `instruction-n` sont exécutées, si elle est fausse, ce sont les instructions `instruction-n+1` à `instruction-m` qui sont exécutées.

2.2. Boucles

La structure d'une boucle `while` est la suivante :

```
while(condition){
instruction-1;
instruction-1;
...
instruction-n;
}
```

Si la condition est vraie, alors les instructions `instruction-1` à `instruction-n` sont exécutées puis la condition est recalculée, si elle est encore vraie, alors les instructions `instruction-1` à `instruction-n` sont exécutées une seconde fois, etc.

Si la condition est fausse, la boucle est terminée, le programme passe à l'instruction suivante. La condition peut être fausse la première fois qu'on la calcule : dans ce cas, les instructions ne sont pas exécutées du tout.

Si elle est fausse la deuxième fois qu'on la calcule, alors les instructions ont été exécutées une fois. Si la condition est fausse la n^{ime} fois qu'on la calcule, alors les instructions ont été exécutées n − 1 fois.

Il se peut que la condition ne devienne jamais fausse et alors le programme exécute les instructions sans jamais s'arrêter. Cela peut être une erreur du programme ou le comportement souhaité par le programmeur. Voici un exemple de boucle while :

```
package hello_world;

public class While {

	public static void main(String[] args) {
		int note;
		Terminal.ecrireString("Entrez votre note: ");
		note = Terminal.lireInt();
		while (note < 0 || note > 20) {
			Terminal.ecrireStringln("Vous vous êtes trompé");
			Terminal.ecrireStringln("La note doit être comprise entre 0 et 20");
			Terminal.ecrireStringln("Veuillez recommencer");
			Terminal.ecrireString("Entrez votre note: ");
			note = Terminal.lireInt();
		}
		Terminal.ecrireStringln("Votre note est " + note);
	}
}
```

La deuxième boucle est la boucle for que l'on utilise lorsque l'on connaît le nombre de tours de boucle à effectuer au moment où l'on commence à exécuter cette boucle.

La boucle for a la structure suivante :

```
for(initialisation;condition;incrémentation){ instruction-1;
...
instruction-n;
}
```

Voici un exemple de boucle for qui affiche 10 lignes à l'écran.

```
for (int i = 1; i < 10; i = i + 1) {
		Terminal.ecrireStringln(i + " kilomètre(s) à pied, ça use, ça use");
		Terminal.ecrireStringln("ça use les souliers");
}
```

Il existe une troisième boucle en java, la boucle do...while. Elle a la structure suivante :

```
do{
instruction-1;
...
instruction-n;
}while(condition);
```

C'est une variante de la boucle while où la condition est calculée à la fin de la boucle, après avoir exécuté les instructions 1 à n. Cela signifie que ces instructions sont toujours calculée au moins une fois.

3. Tableaux

Un tableau est une structure de données qui contient plusieurs valeurs du même type. Les différentes valeurs sont numérotées à partir du numéro 0. S'il y a 10 valeurs, elles sont numérotées de 0 à 9.

0	1	2	3	4	5	6	7	8	9
127	0	2	55	-30	0	125	8	1	45

```
// Il est initialisé avec 10 valeurs
int [] myArray = { 127, 0, 2, 55, -30, 0, 125, 8, 1, 45}
```

Le type d'un tableau est noté avec le type des élément suivi de crochets : int[] pour un tableau contenant des entiers, String[] pour un tableau contenant des chaînes de caractères.

Avant d'utiliser un tableau, il faut le créer avec l'instruction new, en précisant la taille du tableau, c'est à dire le nombre de valeurs qu'il contient. Ce nombre ne changera jamais au cours du programme. new int[12] crée un tableau de 12 cases numérotées de 0 à 11, chaque case contenant un entier. A la création du tableau, chaque case contient 0.

Chaque case du tableau est utilisable comme une variable de type int à laquelle on peut affecter une valeur et dont on peut utiliser la valeur dans un calcul.

```
int[] tab;
tab = new int[12];
tab[0]=145;
tab[1]=37;
Terminal.ecrireIntln(tab[1]*5+1);
```

Les tableaux sont souvent parcourus au moyen d'une boucle for. Voici par exemple un programme qui met dans les cases d'un tableaux les premières puissances de 2 :

```
int[] tab;
tab = new int[8];
tab[0]=1;
for (int i=1;i<8;i++){
tab[i]=tab[i-1]*2;
}
// affichage du tableau
for (int i=0; i<8; i++){
Terminal.ecrireIntln(tab[i]);
}
```

On peut aussi utiliser la syntaxe suivante :

```
for (int i = 0; i < tab.length(); i++) {}
//equivalent mais i dans ce cas prend la valeur du tableau
for (int i : tab) { }
```

On peut obtenir la taille d'un tableau tab au moyen de l'expression `tab.length`. Un tableau à deux dimensions peut être vu comme un tableau ayant des lignes et des colonnes. Chaque case est caractérisée par un numéro de ligne et un numéro de colonne. Par exemple, pour représenter les heures de présence d'une personne à son poste de travail sur une semaine, on peut utiliser un tableau de booléens à 7 colonnes (les 7 jours) et 24 lignes (les 24 heures de la journée). Le booléen sera true si la personne est là à l'heure et au jour correspondant à la case et false si elle n'est pas là.

```
boolean[][] pres = new boolean[7][24];
// les trois premiers jours, la personne est la de 4H00 à 12H00
for (int jour = 0; jour < 3; jour++) {
        for (int heure = 4; heure < 12; heure++) {
                pres[jour][heure] = true;
        }
}
// enseuite, deux jours de repos et deux jours du soir
for (int jour = 5; jour < 7; jour++) {
        for (int heure = 16; heure < 24; heure++) {
                pres[jour][heure] = true;
        }
}
// affichage
for (int heure = 0; heure < 24; heure++) {
        for (int jour = 0; jour < 7; jour++) {
                if (pres[jour][heure]) {
                        Terminal.ecrireString("XXX");
                } else {
                        Terminal.ecrireString(" ");
                }
        }
        Terminal.ecrireStringln("");
}
```

4. Méthodes

Une méthode est un programme distingué soit parce qu'il est utilisé à de multiple reprises dans le programme, soit parce qu'il réalise une tâche spécifique qu'il est plus clair de considérer comme une unité séparée.

Une méthode a des données en entrée qui viennent via des paramètres. Il peut renvoyer une valeur à la fin de son exécution.

On l'écrit avec un entête et un corps. L'entête comprend dans l'ordre : le type de la valeur renvoyée (ou `void` si aucune valeur n'est renvoyée), le nom, la liste des paramètres entre parenthèses. Le corps est un bloc : un suite d'instruction entre accolades. Une instruction spécifique sert à renvoyer le résultat de la méthode, l'instruction `return`.

```
double moyenne(double[] tab) {
        double somme = 0.0;
        for (int i = 0; i < tab.length; i++) {
                somme = somme + tab[i];
        }
        return somme / tab.length;
}
```

5. Réaliser TP Algorithme

Réaliser TP Algorithme

Classes, méthodes et collections

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Classes, méthodes et collections

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:20

Table des matières

1. Introduction

L'approche Orientée Objet, que nous allons aborder dans ce chapitre, consiste à rendre possible dans le langage de programmation la définition d'objets (des livres, des comptes ...) qui ressemblent à ceux du monde réel, c'est à dire caractérisés par un état et un comportement.

L'état d'un compte, pourra être défini par son numéro, le nom de son titulaire, son solde; son comportement est caractérisé par les opérations de depot, de retrait et d'affichage du solde.

Dans nos programmes nous aurons plusieurs objets comptes. Chacun ont un état qui leur est propre, mais ils ont les mêmes caractéristiques : ce sont tous des comptes. En programmation Orientée Objet, nous dirons que ces différents objets comptes sont des objets instances de la classe Compte.

Une classe est un prototype qui définit les variables et les méthodes communes à tous les objets d'un même genre. Une classe est un patron d'objets. Chaque classe définit la façon de créer et de manipuler des Objets de ce type.

A l'inverse, un objet est toujours un exemplaire, une instance d'une classe (son patron). Ainsi, pour faire de la programmation Objet, il faut savoir concevoir des classes, c'est à dire définir des modèles d'objets, et créer des objets à partir de ces classes.

Concevoir une classe, c'est définir :

1. **Les données** caractéristiques des objets de la classe. On appelle ces caractéristiques les *variables d'instance*.
2. **Les actions** que l'on peut effectuer sur les objets de la classe . Ce sont les *méthodes* qui peuvent s'invoquer sur chacun des objets de la classe.

Ainsi, chaque objet crée possèdera :

1. **Un état**, c'est à dire des valeurs particulières pour les variables d'instances de la classe auquel il appartient.
2. **Des méthodes** qui vont agir sur son état.

```java
package poo;

public class Testeur {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Compte martin = new Compte();
		Compte robert = new Compte();

		System.out.println("Mon Solde de Martin " + martin.getSolde());

		martin.credit(1000);
		robert.credit(2000);
		robert.credit(1000);

		System.out.println("Mon Solde de Martin " + martin.getSolde());
		System.out.println("Mon Solde de Robert " + robert.getSolde());
	}

}
```

```java
package poo;

public class Compte {


	private double solde = 0;
	public String nom = "";

	public String getNom() {
		return nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

	public double getSolde() {
		return solde;
	}

	public void setSolde(double solde) {

		this.solde = solde;
	}

	public double credit(double montant) {

		this.solde +=  montant;

		return solde;

	}


}
```

2. Classe et Variables d’instances

Une classe qui définit un type d’objet a la structure suivante :

- Son nom est celui du type que l’on veut créer.
- Elle contient les noms et le type des caractéristiques (les variables d’instances) définissant les objets de ce type.
- Elle contient les méthodes applicables sur les objets de la classe.

Pour définir une classe pour les comptes bancaires on aurait par exemple :

```
public class Compte {
        int solde;
        String titulaire;
        int numero;

        void afficher() {
                Terminal.ecrireStringln("solde: " + this.solde);
        }

        void deposer(int montant) {
                this.solde = this.solde + montant;
        }

        void retirer(int montant) {
                this.solde = this.solde - montant;
        }
}
```

La déclaration d’une **variable d’instance** se fait comme une déclaration de variable locale au main ou à un sous programme : on donne un nom, précédé d’un type. La différence est que cette déclaration se fait **au niveau de la classe** et non à l’intérieur d’un sous programme.

Nous avons dans la classe `Compte` 3 variables d’instance : solde destinée à recevoir des entiers, `titulaire` destinée à recevoir une chaine de caractères et `numero` destinée à recevoir des entiers. Ainsi l’état de chaque objet instance de la classe compte que nous créerons par la suite sera constitué d’une valeur pour chacune de ces trois variables d’instances. Pour chaque objet instance de la classe `Compte` nous pourrons connaitre la valeur de son solde, le nom de son titulaire et le numéro du compte. Ces valeurs seront **propres à chaque objet**.

2.1. Variables d’instances

La différence entre une variable d’instance et une variable de classe (variable statique) en Java.

Variable d’instance	**Variable de classe(variable statique)**
Les variables d’instance sont déclarées dans une classe, mais en dehors d’une méthode, d’un constructeur ou de tout bloc.	Les variables de classe, également appelées variables statiques, sont déclarées avec le mot clé « static » dans une classe, mais en dehors d’une méthode, d’un constructeur ou d’un bloc.
Les variables d’instance sont créées lorsqu’un objet est créé à l’aide du mot clé « new » et détruites lorsque l’objet est détruit.	Les variables statiques sont créées lors de l’exécution du programme et détruites à l’arrêt du programme.
Les variables d’instance sont accessibles directement en appelant le nom de la variable dans la classe. Pourtant, dans les méthodes statiques, elles doivent être appelées en utilisant le nom complet: Object.VariableName.	Les variables statiques sont accessibles en utilisant le nom de classe: ClassName.VariableName.

```
public class MaClasse{

   int value;
   // Variable de classe
   static int nbr = 5;

   public static void main(String args[]){
      //Variable d'instance
      MaClasse obj = new MaClasse();
      obj.value = 6;

      System.out.println("Variable d'instance: "+obj.value);
      System.out.println("Variable de classe: "+MaClasse.nbr);
   }
}
```

2.2. Niveaux de visibilité

La visibilité publique - public

Si vous définissez un membre (attribut, méthode...) public il sera alors accessible de n'importe ou dans le programme. La classe aura donc accès à ce membre, mais aussi n'importe quelle autre classe.

```
public class Demo {

    public int attribute = 10;

    public void doSomething() {
        System.out.println( "Do something" );
    }

}
```

La visibilité protégée - protected

Un membre marqué dans une classe comme étant protégé (mot clé `protected`) peut être manipulé :

- Dans la classe qui définit ce membre,
- Dans les classes qui dérivent de la classe considérée,
- Et dans toutes les classes (et les types) définies dans le même package que celle qui définit le membre protégé

```
package fr.koor.sample;

import java.awt.Point;

class Shape {
    protected Point center = new Point();       // Essayez avec un private => Ko
}

class Circle extends Shape {
    public Circle() {
        super();
        System.out.println( "Center == " + this.center );
    }
}

public class Sample {

    public static void main( String[] args ) {
        Circle c = new Circle();
        // TODO
    }

}
```

La visibilité « package private »

En l'absence de mot clé pour spécifier la visibilité, un membre sera considéré comme étant « package private » (on peut simplifier en parlant de visibilité package). C'est donc le mode de visibilité par défaut (absence de mot clé). Cela veut dire que le membre sera accessible par tout code Java localisé dans le même package. Voici un exemple d'utilisation.

```
package fr.koor.sample;

public class TestVisibility {
    public int attr1 = 10;
    protected int attr2 = 20;
    int attr3 = 30;             // Visibilité package private !
    private int attr4 = 40;

}
```

La visibilité privée - private

Dans ce dernier cas, seule la classe ayant définit le membre pourra y accéder.

La visibilité des vos classes

Vous pouvez aussi définir des niveaux de visibilités sur vos définitions de classes. Différents cas sont à considérer.

Les classes publiques

Pour rappel, vous pouvez définir qu'une seule classe publique par fichier. Cette classe publique doit avoir très exactement le même nom que celui du fichier (sans l'extension, bien entendu) et de manière « case sensitive ». Une classe publique est accessible en tout point de votre programme Java. Je ne reviendrais pas plus sur cette possibilité, on en parle déjà depuis le début de ce tutoriel Java.

Les classes « package private »

Un fichier de code Java peut contenir bien plus qu'une seule définition de classe. Mais, outre la classe publique, les autres seront considérés comme des classes restreintes au package dans lequel est placé le fichier. Elles ne devront porter aucun qualificateur de visibilité.

```java
package fr.koor.sample;

// Classe visible uniquement dans le package fr.koor.sample.
class PrivateClass {

    public PrivateClass() {
        System.out.println( "PrivateClass constructor" );
    }

}

// Classe publique visible dans tout le projet.
public class Sample {

    public Sample() {
        PrivateClass obj = new PrivateClass();
        // TODO
    }

}
```

3. Les méthodes

Nous n'allons pas dans ce paragraphe décrire dans le détail la définition des méthodes d'objets, mais nous nous contentons pour l'instant des remarques suivantes : une classe définissant un type d'objets comportera autant de méthodes qu'il y a d'opérations utiles sur les objets de la classe.

La définition d'une méthode d'objet (ou d'instance) ressemble à la définition d'un sous pro- gramme : un type de retour, un nom, une liste d'arguments précédés de leur type. Ce qui fait d'elle une méthode d'objet est qu'elle ne comporte pas le mot clé `static`. Ceci (plus le fait que les méthodes sont dans la classe `Compte`) indique que la méthode va pouvoir être invoquée (appelée) sur n'importe quel objet de type `Compte` et modifier son état (le contenu de ses variables d'instances).

Ceci a aussi des conséquences sur le code de la méthode comme par exemple l'apparition du mot clé `this`, sur lequel nous reviendrons lorsque nous saurons comment invoquer une méthode sur un objet.

Une fois définie une classe d'objets, on peut utiliser le nom de la classe comme un nouveau type : déclaration de variables, d'arguments de sous programmes On pourra de plus appliquer sur les objets de ce type toutes les méthodes de la classe.

Si la classe `Compte` est dans votre répertoire de travail, vous pouvez maintenant écrire une autre classe, par exemple `test` qui, dans son main, déclare une variable de type `Compte` :

```
public class Test {
    public static void main (String [] arguments){
       Compte c1 = new Compte();
    }
}
```

Comme pour les tableaux, une variable référençant un objet de la classe `Compte`, doit recevoir une valeur, soit par une affectation d'une valeur déjà existante, soit en créant une nouvelle valeur avec `new` avant d'être utilisée. On peut séparer la déclaration et l'initialisation en deux instructions :

```
public class Test {
    public static void main (String [] arguments){
       Compte c1;
       c1 = new Compte();
    }
}
```

Après l'exécution de `c1 = new Compte();` chaque variable d'instance de `c1` a une valeur par défaut. Cette valeur est 0 pour `solde` et `numero`, et `null` pour `titulaire`.

La classe `Compte` définit la forme commune à tous les comptes. Toutes les variable de type `Compte` auront donc en commun cette forme : un solde, un titulaire et un numéro. En revanche, elles pourront représenter des comptes différents.

Accéder aux valeurs des variables d'instance

Comment connaître le solde du compte `c1` ? Ceci se fait par l'**opérateur noté par un point** :

```
public class Test {
        public static void main(String[] arguments) {
                Compte c1 = new Compte();
                Terminal.ecrireInt(c1.solde);

        }
}
```

La dernière instruction à pour effet d'afficher à l'écran la valeur de la variable d'instance `solde` de `c1`, c'est à dire l'entier 0. Comme le champ `solde` est de type `int`, l'expression `c1.solde` peut s'utiliser partout où un entier est utilisable :

```
public class Test {
        public static void main(String[] arguments) {
                Compte c1 = new Compte();
                int x;
                int[] tab = { 2, 4, 6 };
                tab[c1.solde] = 3;
                tab[1] = c1.numero;
                x = d1.solde + 34 / (d1.numero + 4);
        }
}
```

Modifier les valeurs des variables d'instance

Chaque variable d'instance se comporte comme une variable. On peut donc lui affecter une nouvelle valeur :

```java
public class Test {
        public static void main(String[] arguments) {
                Compte c1 = new Compte();
                Compte c2 = new Compte();
                c1.solde = 100;
                c1.numero = 218;
                c1.titulaire = "Dupont";
                c2.solde = 200;
                c2.numero = 111;
                c2.titulaire = "Durand";
                Terminal.ecrireStringln("valeur de c1:" + c1.solde + " , " + c1.titulaire + " ," + c1.numero);
                Terminal.ecrireStringln("valeur de c2:" + c2.solde + " , " + c2.titulaire + " ," + c2.numero);
        }
}
```

`c1` représente maintenant le compte numero 218 appartenant à Dupont et ayant un solde de 100 euros. et `c2` le compte numero 111 appartenant à Durand et ayant un solde de 200 euros.

Affectation entre variables référençant des objets

l'affectation entre variables de types Compte est possible, puisqu'elles sont du même type, mais le même phenomène qu'avec les tableaux se produit : les 2 variables référencent le même objet et toute modification de l'une modifie aussi l'autre :

```java
public class TestBis {
        public static void main(String[] arguments) {
                Compte c1 = new Compte();
                Compte c2 = new Compte();
                c1.solde = 100;
                c1.numero = 218;
                c1.titulaire = "Dupont";
                c2 = c1;
                c2.solde = 60;
                Terminal.ecrireStringln("valeur de c1:" + c1.solde + " , " + c1.titulaire + ", " + c1.numero);
                Terminal.ecrireStringln("valeur de c2:" + c2.solde + " , " + c2.titulaire + ", " + c2.numero);
        }
}
```

Trace d'exécution :

```
%> java TestBis

valeur de c1: 60 , Dupont, 218 valeur de c2: 60 , Dupont, 218
```

Invoquer les méthodes sur les objets.

Une classe contient des variables d'instances et des méthodes. Chaque objet instance de cette classe aura son propre état, c'est à dire ses propres valeurs pour les variables d'instances. On pourra aussi invoquer sur lui chaque méthode non statique de la classe. Comment invoque-t-on une méthode sur un objet ?

Pour invoquer la méthode afficher() sur un objet c1 de la classe Compte il faut écrire :

```
c1.afficher();
```

Ainsi, les méthodes d'objets (ou méthodes non statiques) s'utilisent par invocation sur les objets de la classe dans lesquelles elles sont définies. l'objet sur lequel on l'invoque ne fait pas partie de la liste des arguments de la méthode. Nous l'appellerons l'objet courant.

3.1. méthodes non statiques

Dans une classe définissant un type d'objet, on définit l'état caractéristiques des objets de la classe (les variables d'instances) et les méthodes capables d'agir sur l'état des objets (méthodes non statiques). Pour utiliser ces methodes sur un objet x donné, on ne met pas *x* dans la liste des arguments de la méthode. On utilisera la classe en déclarant des objets instances de cette classe. Sur chacun de ces objets, la notation pointée permettra d'accéder à l'état de l'objet (la valeur de ses variables d'ins- tances) ou de lui appliquer une des méthodes de la classe dont il est une instance.

Par exemple, si c1 est un objet instance de la classe Compte c1.titulaire permet d'accéder au titulaire de ce compte, et c1.deposer(800) permet d'invoquer la méthode deposer sur c1.

3.2. TP Compte

Réaliser TP Compte

4. Surcharge

Dans une classe on peut avoir plusieurs méthodes différentes avec le même nom. Ces méthodes sont différentes parce qu'elles ne comportent pas les mêmes instructions. Pour pouvoir savoir laquelle des méthodes qui portent le même nom est appelée lors d'une invocation de méthode, il faut que les paramètres soient différents par le type et/ou par le nombre de telle sorte qu'il n'y a pas de doute possible : une seule méthode correspond aux valeurs données entre parenthèse dans l'invocation de méthode.

Dans ce PAP Java nous utiliserons uniquement la surcharge de la methode toString() qui permet de renvoyer les informations sous forme string d'in objet.

```java
// methode surcharge Override
        public String toString() {
             return this.name+" "+ this.lastname;
        }
```

5. List et Arraylist (collections)

List et ArrayList sont les membres du framework **Collection**.

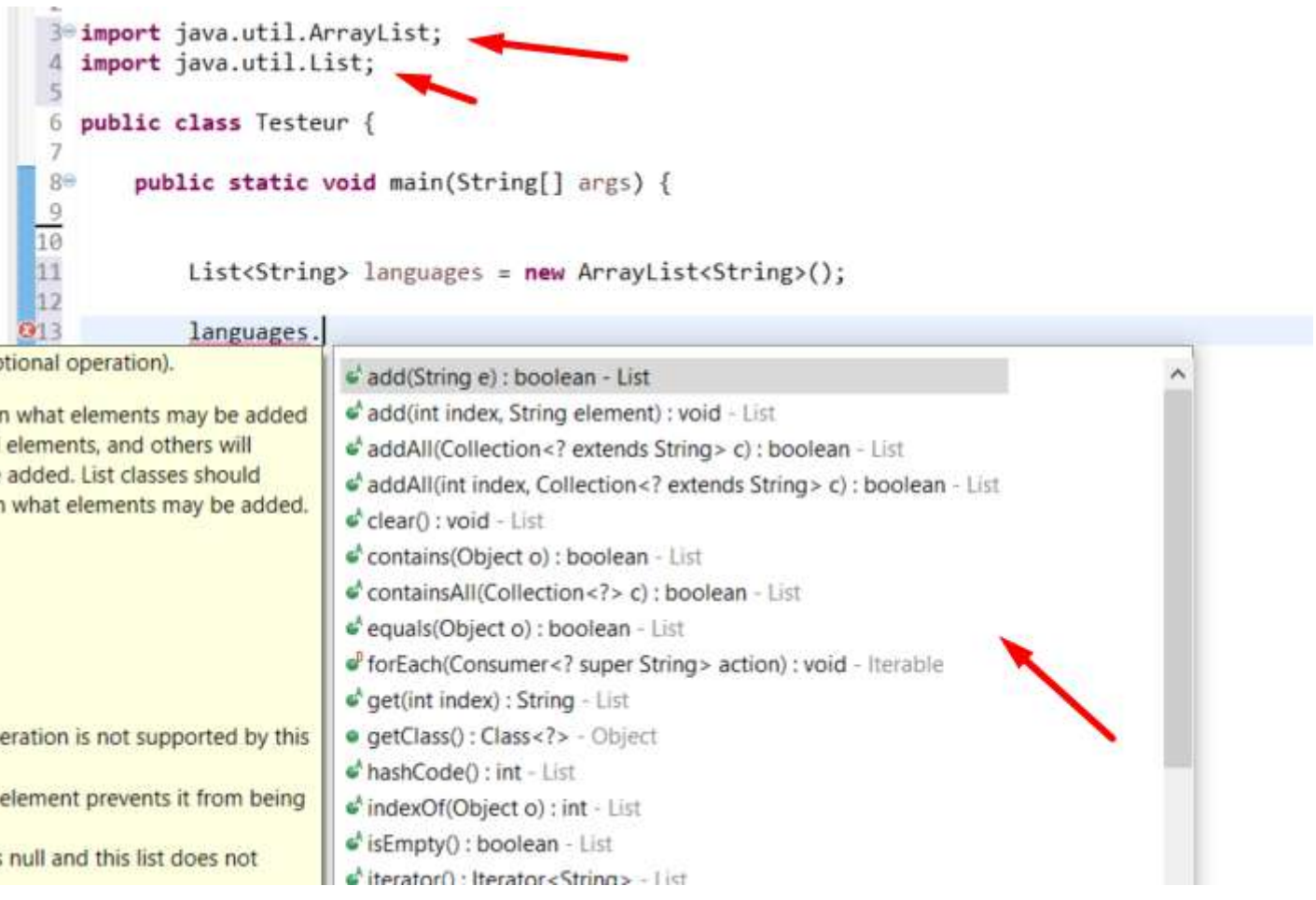
La liste est une collection d'éléments dans une séquence où chaque élément est un objet et les éléments sont accessibles par leur position (index).

ArrayList crée un tableau dynamique d'objets dont la taille augmente ou diminue à chaque fois que nécessaire. La principale différence entre List et ArrayList est que List est une interface et ArrayList est une classe.

Exemple pour stocker des personnes dans le constructeur de la class Compte :

```
private List<Personne> personnes;

        public Compte(String name) {
                this.name = name;
                this.personnes = new ArrayList<Personne>();
        }
```

Remarque : `List` et `ArrayList` sont associé à la class `java.util`. L'objet ArrayList dispose de nombreuses méthodes pour ajouter une valeur, supprimer, etc.



5.1. TP ArrayList

TP ArrayList

Constructeurs et héritage

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Constructeurs et héritage

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:21

Table des matières

1. Le constructeur par défaut

Que se passe-t-il lorsque nous écrivons : `Date d1= new Date()` ?

L'opérateur `new` réserve l'espace mémoire nécessaire pour l'objet `d1` et initialise les données avec des valeurs par défaut. Une variable d'instance de type `int` recevra `0`, une de type `boolean` recevra `false`, une de type `char` recevra `'\0'` et les variables d'instances d'un type objet recevra `null`. Nous expliquerons ce qu'est `null` dans le prochain chapitre.

L'opérateur `new` réalise cela en s'aidant d'un *constructeur* de la class `Date`. En effet, lorsqu'on écrit `Date()`, on appelle une sorte de méthode qui porte le même nom que la `classe`. Dans notre exemple, on voit que le constructeur Date qui est appelé n'a pas d'arguments.

Un constructeur ressemble à une méthode qui porterai le même nom que la classe, mais ce n'est pas une méthode : la seule façon de l'invoquer consiste à employer le mot clé `new` suivi de son nom, c'est à dire du nom de la classe. Ceci signifie qu'il s'exécute avant toute autre action sur l'objet, lors de la création de l'objet.

La classe `Date` ne contient pas explicitement de définition de constructeur. Et pourtant, nous pouvons, lors de la création d'un objet, y faire référence après l'opérateur new. Cela signifie que Java fournit pour chaque classe définie, un constructeur par défaut. Ce constructeur par défaut :

- a le même nom que la classe et
- n'a pas d'argument.

Le constructeur par défaut ne fait pratiquement rien. Voilà a quoi il pourrait ressembler :

```
public Date(){
}
```

Le point intéressant avec les constructeurs est que nous pouvons les définir nous mêmes. Nous avons ainsi un moyen d'intervenir au milieu de `new`, d'intervenir lors de la création des objets, donc avant toute autre action sur l'objet.

Autrement dit, en écrivons nos propres constructeurs, nous avons le moyen, en tant que concepteur d'une classe, d'intervenir pour préparer l'objet à être utilisé, avant toute autre personne utilisatrice des objets.

Sans définir nos propres constructeurs de `Date`, les objets de types `Date` commencent mal leur vie : il naissent avec 0,0,0 qui ne représente pas une date correcte. En effet, 0 ne correspond pas à un jour, ni à un mois. C'est pourquoi une bonne conception de la classe `Date` comportera des définitions de constructeurs.

Attention : Dès que nous définissons un constructeur pour une classe, le constructeur par défaut n'existe plus.

Un constructeur se définit comme une méthode sauf que :

1. Le nom d'un constructeur est toujours celui de la classe.
2. Un constructeur n'a jamais de type de retour.

Dans une classe, on peut définir autant de constructeurs que l'on veut, du moment qu'ils se différencient par leur nombre (ou le type) d'arguments . Autrement dit, on peut surcharger les constructeurs.

Nous pourrions par exemple, pour notre classe `Date`, définir un constructeur sans arguments qui initialise les dates à 1,1,1 (qui est une date correcte) :

```
public class Date {
		int jour;
		int mois;
		int annee;

		public Date() {
			this.jour = 1;
			this.mois = 1;
			this.annee = 1;
		}
		// ....
	}
```

Maintenant, toute invocation de `new Date()` exécutera ce constructeur.

Il peut aussi être utile de définir un constructeur qui initialise une date avec des valeurs données.

Pour cela, il suffit d'écrire un constructeur avec 3 arguments qui seront les valeurs respectives des champs. Si les valeurs d'entrée ne représentent pas une date correcte, nous levons une erreur.

1.1. Exemple classe Date

```java
package constructeurs;

public class Date {
	// ---Les variables d’instances ---
	int jour;
	int mois;

	int annee;

	// ---Les constructeurs ---
	public Date() {
		this.jour = 1;
		this.mois = 1;
		this.annee = 1;
	}

	public Date(int j, int m, int a) {
		if (m > 0 && m < 13 && j <= longueur(m, a)) {
			this.jour = j;
			this.mois = m;
			this.annee = a;
		} else {
			throw new ErreurDate();
		}
	}

	// ---Les methodes ---
	public void afficherDate() {
		Terminal.ecrireStringln(this.jour + " , " + this.mois + " , " + this.annee);
	}

	public int getAnnee() {
		return this.annee;
	}

	public void setAnnee(int aa) {
		this.annee = aa;
	}

	public void passerAuLendemain() {
		if (this.jour < longueur(this.mois, this.annee)) {
			this.jour = this.jour + 1;
		} else if (this.mois == 12) {
			this.jour = 1;
			this.annee = this.annee + 1;
			this.mois = 1;
		} else {
			this.jour = 1;
			this.mois = this.mois + 1;
		}
	}

	public static boolean bissextile(int annee) {
		return ((annee % 4 == 0) && (!(annee % 100 == 0) || annee % 400 == 0));
	}

	public static int longueur(int m, int a) {
		if (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12) {
			return 31;
		}

		else if (m == 2) {
			if (bissextile(a)) {
				return 29;
			} else {
				return 28;
			}
		} else {
			return 30;
		}
	}
}

class ErreurDate extends Error {
}
```

Testons cette class :

```java
package constructeurs;

public class TesteDate {

	public static void main(String[] arguments) {
		Date d1 = new Date();
		Date d2 = new Date(2, 12, 2000);
		d1.afficherDate();
		d2.afficherDate();
		d2.passerAuLendemain();
		d2.afficherDate();

	}
}
```

L'éxécution de `TesteDate` donne :

1 , 1 , 1
2 , 12 , 2000
3 , 12 , 2000

Répétons encore une fois que lorsqu'on définit ses propres constructeurs, le constructeur par défaut n'existe plus. En conséquence, si vous écrivez des constructeurs qui ont des arguments, et que vous voulez un constructeur sans argument, vous devez l'écrire vous-même.

1.2. Rappel

- Un constructeur :
  - est un code qui s’éxécute au moment de la création de l’objet (avec new),
  - porte le même nom que la classe,
  - ne peut pas avoir de type de retour,
  - peut avoir ou ne pas avoir d’arguments,
  - sert à initialiser l’état de chaque objet crée.
- Si on ne définit aucun constructeur, il en existe un par défaut, qui n’a pas d’argument.
- Le constructeur par défaut n’existe plus dès que l’on définit un constructeur même si on ne définit pas de constructeur sans argument.
- 2 constructeurs doivent avoir des listes d’arguments différentes.

1.3. TP Cocktail

Réaliser TP Cocktail

2. Héritage

L'héritage est un concept-clé de la programmation objet. Il permet de définir une classe en ajoutant des composants supplémentaires à une classe existante. Ces composants sont notamment des attributs et des méthodes. Par ailleurs, il est possible de redéfinir une méthode, c'est-à-dire de conserver le même nom et les mêmes paramètres, mais de changer les instructions de la méthode.

Dans ce chapitre, nous allons commencer l'étude de l'héritage en voyant comment utiliser l'héritage pour définir de nouvelles classes.

Nous allons réutiliser ici l'exemple des comptes bancaires donné dans le cours sur les classes.

Sans redonner tout le code, rappelons les composants de la classe.

- Attributs : numéro (int), titulaire (String) et solde (double).
- Constructeur : `Compte(String tit, int num)`
- Accesseurs : `getSolde`, `getTitulaire`, `getNumero`
- Modificateurs : `deposer`, `retirer`
- Méthode de service : `toString`

Lorsqu'on va écrire une classe qui hérite de la classe `Compte`, nécessairement, il y aura tous les attributs et toutes les méthodes de la classe. Le constructeur n'est pas hérité, il faut en écrire un autre. Certaines méthodes peuvent être redéfinies.

Nous allons définir une classe par héritage depuis `Compte`. On dira que la nouvelle classe est une *sous-classe* ou une *classe fille* de Compte et que Compte est la *super-classe* ou la *classe mère* de la nouvelle classe.

Nous allons définir une nouvelle classe de compte dans laquelle on vérifiera que le compte ne dépasse pas un découvert fixé, ce découvert pouvant être différent d'un compte à un autre. Pour ce faire, il faut ajouter un attribut `decouvertAutorise`, un accesseur `getDecouvert` et un modificateur `setDecouvert`. Par ailleurs, il faudra redéfinir la méthode `retirer` pour tenir compte du découvert autorisé.

2.1. Exemple Classe Compte

```java
package tp02_compte;

public class CompteDecouvert extends Compte {
	private double decouvertAutorise = 500;

	public CompteDecouvert(String tit, int num) {
		super(tit, num);
	}

	public double getDecouvert() {
		return decouvertAutorise;
	}

	public void setDecouvert(double decouv) {
		if (decouv < 0) {
			throw new IllegalArgumentException("Un montant ne peut pas être négatif");
		} else {
			decouvertAutorise = decouv;
		}
	}

	public void retirer(double montant) {
		if (montant < 0) {
			throw new IllegalArgumentException("Un montant ne peut pas être négatif");
		} else if (solde - montant < -decouvertAutorise) {
			throw new IllegalArgumentException("Dépassement du découvert autorisé");
		} else {
			solde = solde - montant;
		}
	}
}
```

Le fait que l'on utilise l'héritage apparaît sur la première ligne avec la clause `extends Compte`.

Cela signifie que la classe `CompteDecouvert` va hériter des attributs et méthodes de Compte. Dans le corps de la classe, l'attribut `decouvertAutorise` ainsi que sont accesseur et son modificateur sont définis. Il y a également un constructeur. Le constructeur de la sous-classe doit nécessairement appeler le constructeur de la super-classe en utilisant le mot-clé `super`.

Ce mot-clé est suivi des paramètres qu'il faut donner au constructeur de la super-classe. Il faut nécessairement que cet appel apparaisse comme première instruction du constructeur de la sous-classe, avant toute autre instruction. Cet appel au constructeur de la super-classe ne correspond pas à un `new` : il ne crée pas un nouvel objet de la classe `Compte`, mais il contribue à créer l'objet instance de `CompteDecouvert`. C'est-à-dire que les instructions du constructeur de la super-classe `Compte` sont exécutées sur un objet instance de `CompteDecouvert`.

La méthode `retirer` est redéfinie pour empêcher de passer au-delà du découvert autorisé à l'occasion d'un retrait. Le code est différent. On voit que cette méthode a besoin d'accéder à l'attribut `solde` de la classe `Compte`. Or celui-ci avait été déclaré `private`, ce qui signifie qu'on ne peut l'utiliser que dans la classe Compte. Au moment de la compilation, cela provoque l'erreur suivante :

```
error: solde has private access in Compte
```

2.2. TP Héritage

Réaliser TP Héritage

Interface et class abstraite

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Interface et class abstraite

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:21

Description

Interface et class abstraite

Table des matières

1. L’abstraction

L’abstraction est le processus consistant à masquer l’implémentation et à ne fournir que les détails essentiels à l’utilisateur. L’abstraction en Java est réalisée via des classes abstraites et des interfaces. Les classes abstraites et les interfaces ont quelques points communs, mais il existe des différences majeures entre elles. Examinons d’abord certaines des choses qui sont similaires entre les deux.

- Comme indiqué ci-dessus, la classe abstraite et l’interface sont utilisées pour l’abstraction.
- Les classes abstraites et les interfaces `ne peuvent pas être instanciées`, c’est-à-dire que nous ne pouvons pas créer d’objet pour elles.
- Les sous-classes doivent remplacer les méthodes abstraites définies dans la classe abstraite ou l’interface.

Les points ci-dessus résument à peu près les similitudes entre les deux. Examinons maintenant certaines des principales différences entre eux.

2. Différences

Différences entre la classe abstraite et l’interface

Classe abstraite	Interface
Le mot-clé `abstract` en Java est utilisé pour créer ou déclarer une classe abstraite.	En Java, le mot-clé `interface` est utilisé pour créer ou déclarer une nouvelle interface.
Une classe peut hériter des propriétés et méthodes d’une classe abstraite en utilisant le mot-clé `extends`.	Pour implémenter une Interface en Java, on peut utiliser le mot-clé `implements`.
Une classe abstraite peut avoir des méthodes abstraites ou non abstraites définies en elle. Les méthodes abstraites sont celles pour lesquelles aucune implémentation n’est fournie.	Une interface ne peut contenir que des méthodes abstraites. Nous ne pouvons fournir que la définition de la méthode mais pas son implémentation. Après Java 8, nous pouvons également avoir des méthodes par défaut et statiques dans les interfaces.
Une classe abstraite peut contenir des variables finales ou non finales (attributs de classe). Il peut également contenir des attributs statiques ou non statiques.	Une interface ne peut contenir que des membres statiques et finaux, et aucun autre type de membre n’est autorisé.
Une classe abstraite peut implémenter une interface et implémenter les méthodes de l’interface.	Une interface ne peut étendre aucune autre classe et ne peut pas surcharger ou implémenter des méthodes de classe abstraite.
Une classe abstraite peut étendre d’autres classes et peut également implémenter des interfaces.	Comme discuté dans le point précédent, les interfaces ne peuvent pas étendre d’autres classes. Mais il n’y a aucune restriction dans l’implémentation d’une interface.
Java ne prend pas en charge les héritages multiples via des classes. Les classes abstraites, comme toute autre classe, ne prennent pas en charge les héritages multiples.	La prise en charge des héritages multiples en Java est fournie via les interfaces. C’est parce que les interfaces fournissent une abstraction complète.
Les membres ou attributs de classe abstraits peuvent être privés, protégés ou publics.	Les attributs ou les membres d’une Interface sont toujours publics.

3. Cas d'utilisation

Les classes abstraites peuvent fournir une abstraction partielle ou complète. Les interfaces, en revanche, fournissent toujours une abstraction complète. Une classe parent abstraite peut être créée pour quelques classes qui ont des fonctionnalités communes. Les classes abstraites sont également privilégiées si vous souhaitez plus de liberté d’action.

Les interfaces sont privilégiées lorsque l’on veut définir une structure de base. Le programmeur peut alors construire n’importe quoi avec cette structure. Les interfaces prennent également en charge les héritages multiples. Ainsi, une seule classe peut implémenter plusieurs interfaces.

Dans l’ensemble, c’est une question de choix et de la tâche qui doit être accomplie. La classe abstraite et l’interface conviennent à des fins différentes et doivent être utilisées en conséquence.

3.1. Classe abstraite

Créons une classe abstraite et créons des classes enfants qui l'étendent pour comprendre la classe abstraite et ses fonctionnalités.

```
abstract class Bell {
        protected String sound;

        Bell() {
                this.sound = "ting";
        }

        // Abstract Method
        abstract public void ring();

        // Non-Abstract Methods
        public void increaseVolume() {
                System.out.println("Increasing Volume");
        }

        public void decreaseVolume() {
                System.out.println("Decreasing Volume");
        }
}

class SchoolBell extends Bell {
        @Override
        public void ring() {
                System.out.println("Ringing the School bell: " + sound);
        }
}

class ChruchBell extends Bell {
        @Override
        public void ring() {
                System.out.println("Ringing the Chruch Bell: " + sound);
        }
}

public class AbstractClassDemo {
        public static void main(String[] args) {
                SchoolBell sb = new SchoolBell();
                ChruchBell cb = new ChruchBell();

                // Using the overridden methods
                sb.ring();
                cb.ring();

                // Using the non-abstract methods of Bell class
                sb.increaseVolume();
                cb.decreaseVolume();
        }
}
```

Production:

```
Ringing the School bell: ting
Ringing the Chruch Bell: ting
Increasing Volume
Decreasing Volume
```

3.2. Interface

Reproduisons le même scénario à l'aide d'interfaces. Nous ne pouvons plus définir de méthodes non abstraites dans l'interface. Interface est le bon choix si les classes ne souhaitent pas une implémentation commune des méthodes `increaseVolume()` et `decreaseVolume()`.

```
interface Bell
{
        String sound = "ting";
        //only abstract methods allowed in interface
        public void ring();
        public void increaseVolume();
        public void decreaseVolume();
}
class SchoolBell implements Bell
{
        public void ring()
        {
                System.out.println("Ringing the School bell: " + sound);
        }
        @Override
        public void increaseVolume()
        {
                System.out.println("Increasing Volume of School Bell");
        }
        @Override
        public void decreaseVolume()
    {
                System.out.println("Decreasing Volume of School Bell");
        }
}
class ChruchBell implements Bell
{
        public void ring()
        {
                System.out.println("Ringing the Chruch Bell: " + sound);
        }
        @Override
        public void increaseVolume()
        {
                System.out.println("Increasing Volume of Chruch Bell");
        }
        @Override
        public void decreaseVolume()
        {
                System.out.println("Decreasing Volume of Chruch Bell");
        }
}
public class InterfaceDemo
{
        public static void main(String[] args)
        {
                SchoolBell sb = new SchoolBell();
                ChruchBell cb = new ChruchBell();

                //Using the overridden methods
                sb.ring();
                cb.ring();

                //Using the non-abstract methods of Bell class
                sb.increaseVolume();
                cb.decreaseVolume();
        }
}
```

Production:

```
Ringing the School bell: ting
Ringing the Chruch Bell: ting
Increasing Volume of School Bell
Decreasing Volume of Chruch Bell
```

3.3. Implémentant interface

Comme discuté dans la section précédente, nous pouvons implémenter les méthodes d'une interface dans une classe abstraite. Le code suivant le démontre.

```
interface Bell
{
        String sound = "ting";
        //only abstract methods allowed in interface
        public void ring();
        public void increaseVolume();
        public void decreaseVolume();
}
abstract class AbstractBell implements Bell
{
        public void increaseVolume()
        {
                System.out.println("Increasing Volume");
        }
        public void decreaseVolume()
        {
                System.out.println("Decreasing Volume");
        }
}
```

L'abstraction est l'un des concepts les plus fondamentaux utilisés dans la programmation orientée objet. L'abstraction est utilisée pour masquer l'implémentation et ne fournir que le minimum de détails essentiels à l'utilisateur. En Java, l'abstraction est réalisée en utilisant des classes abstraites ou des interfaces. Une différence majeure entre les deux est que les classes abstraites peuvent également fournir une abstraction partielle, tandis qu'Interface fournira toujours une abstraction complète. Dans ce didacticiel, nous avons discuté de certaines des principales différences entre les deux.

4. TP

Réaliser TP Seigneur des anneaux

Interfaces graphiques avec SWING

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: Interfaces graphiques avec SWING

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:21

Description

Swing fait partie de la bibliothèque Java Foundation Classes (JFC). C'est une API dont le but est similaire à celui de l'API AWT mais dont les modes de fonctionnement et d'utilisation sont complètement différents. Swing a été intégré au JDK depuis sa version 1.2.

Table des matières

1. Présentation de Swing

Swing propose de nombreux composants dont certains possèdent des fonctions étendues, une utilisation des mécanismes de gestion d'événements performants (ceux introduits par le JDK 1.1) et une apparence modifiable à la volée (une interface graphique qui emploie le style du système d'exploitation Windows ou Motif ou un nouveau style spécifique à Java nommé Metal).

Tous les composants Swing possèdent les caractéristiques suivantes :

- ce sont des beans
- ce sont des composants légers (pas de partie native) hormis quelques exceptions.
- leurs bords peuvent être changés

Swing contient plusieurs packages :

javax.swing	package principal : il contient les interfaces, les principaux composants, les modèles par défaut
javax.swing.border	Classes représentant les bordures
javax.swing.colorchooser	Classes définissant un composant pour la sélection de couleurs
javax.swing.event	Classes et interfaces pour les événements spécifiques à Swing. Les autres événements sont ceux d'AWT (java.awt.event)
javax.swing.filechooser	Classes définissant un composant pour la sélection de fichiers
javax.swing.plaf	Classes et interfaces génériques pour gérer l'apparence
javax.swing.plaf.basic	Classes et interfaces de base pour gérer l'apparence
javax.swing.plaf.metal	Classes et interfaces pour définir l'apparence Metal qui est l'apparence par défaut
javax.swing.table	Classes définissant un composant pour la présentation de données sous forme de tableau
javax.swing.text	Classes et interfaces de bases pour les composants manipulant du texte
javax.swing.text.html	Classes permettant le support du format HTML
javax.swing.text.html.parser	Classes permettant d'analyser des données au format HTML
javax.swing.text.rtf	Classes permettant le support du format RTF
javax.swing.tree	Classes définissant un composant pour la présentation de données sous forme d'arbre
javax.swing.undo	Classes permettant d'implémenter les fonctions annuler/refaire

2. Créer une fenêtre

JFrame est l'équivalent de la classe Frame de l'AWT : les principales différences sont l'utilisation du double buffering qui améliore les rafraichissements et l'utilisation d'un panneau de contenu (contentPane) pour insérer des composants (ils ne sont plus insérés sans le JFrame mais dans l'objet contentPane qui lui est associé). Elle représente une fenêtre principale qui possède un titre, une taille modifiable et éventuellement un menu.

La classe possède plusieurs constructeurs :

Constructeur	Rôle
JFrame()	
JFrame(String)	Création d'une instance en précisant le titre

Par défaut, la fenêtre créée n'est pas visible. La méthode setVisible() permet de l'afficher.

```
import javax.swing.*;

public class TestJFrame1 {

  public static void main(String argv[]) {
    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    f.setVisible(true);
  }
}
```

2.1. Centrer une fenêtre

Par défaut, une JFrame est affichée dans le coin supérieur gauche de l'écran. Pour la centrer dans l'écran, il faut procéder comme pour une Frame : déterminer la position de la Frame en fonction de sa dimension et de celle de l'écran et utiliser la méthode setLocation() pour affecter cette position.

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

public class TestJFrame5 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);

    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    f.setLocation(dim.width/2 - f.getWidth()/2, dim.height/2 - f.getHeight()/2);

    f.setVisible(true);
  }
}
```

3. Image

Pour afficher une image dans une fenêtre, il y a plusieurs solutions.

En utilisant setContentPane :

```
package swing;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Testimg {


        public static void main(String[] args) throws Exception {
                JFrame f = new JFrame("ma fenetre");
            f.setSize(300,300);


                //Background
                f.setContentPane(new JLabel(new ImageIcon("Duke.gif")));

            f.setVisible(true);
        }
}
```

Avec l'option JLabel qui est capable d'afficher du texte mais aussi une image.

```
import java.awt.BorderLayout;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class MonApp  extends JFrame {

  private static final long serialVersionUID = 1L;

  public MonApp(String titre) {
    super(titre);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    init();
  }

  private void init()
  {
    JLabel label = new JLabel(new ImageIcon("Duke.gif") );
    this.add(label, BorderLayout.CENTER);
    this.pack();
  }

  public static void main(String[] args) {
    MonApp app = new MonApp("Afficher image");
    app.setVisible(true);
  }
}
```

3.1. Icône

La méthode setIconImage() permet de modifier l'icône de la JFrame.


```
import javax.swing.*;

public class TestJFrame4 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300,100);
    JButton b =new JButton("Mon bouton");
    f.getContentPane().add(b);
    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    ImageIcon image = new ImageIcon("book.gif");
    f.setIconImage(image.getImage());
    f.setVisible(true);

  }
}
```

3.2. Opacité

Il est possible de jouer sur l'opacité d'une image, exemple :

```
package swing;
import java.awt.AlphaComposite;
import java.awt.Graphics;

import java.awt.Graphics2D;
import java.awt.Image;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.Timer;

import heroicfantasy.Scene;

public class FadeIn extends JPanel implements ActionListener {

    Image imagem;
    Timer timer;
    private float alpha = 0f;

    public FadeIn() {
        imagem = new ImageIcon(Scene.assetPath + "heros\\frodon.png").getImage();
        timer = new Timer(100, this);
        timer.start();
    }
// here you define alpha 0f to 1f
    public FadeIn(float alpha) {
        imagem = new ImageIcon(Scene.assetPath + "heros\\frodon.png").getImage();
        this.alpha = alpha;

    }
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D) g;

        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
                                                    alpha));
        g2d.drawImage(imagem, 0, 0, null);

    }

    public static void main(String[] args) {

        JFrame frame = new JFrame("Fade out");
        frame.add(new FadeIn());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(420, 330);
       // frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }


    public void actionPerformed(ActionEvent e) {
        alpha += 0.05f;
        if (alpha >1) {
            alpha = 1;
            timer.stop();
        }
        repaint();
    }
}
```

4. Boutons

Tous les boutons peuvent afficher du texte et/ou une image.

Il est possible de préciser une image différente lors du passage de la souris sur le composant et lors de l'enfoncement du bouton : dans ce cas, il faut créer trois images pour chacun des états (normal, enfoncé et survolé). L'image normale est associée au bouton grâce au constructeur, l'image enfoncée grâce à la méthode setPressedIcon() et l'image lors d'un survol grâce à la méthode setRolloverIcon(). Il suffit enfin d'appeler la méthode setRolloverEnable() avec en paramètre la valeur true.

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

public class swing4 extends JFrame {

   public swing4() {
     super("titre de l'application");

     WindowListener l = new WindowAdapter() {
       public void windowClosing(WindowEvent e){
         System.exit(0);
       }
     };
     addWindowListener(l);

     ImageIcon imageNormale = new ImageIcon("arrow.gif");
     ImageIcon imagePassage =  new ImageIcon("arrowr.gif");
     ImageIcon imageEnfoncee = new ImageIcon("arrowy.gif");

     JButton bouton = new JButton("Mon bouton",imageNormale);
     bouton.setPressedIcon(imageEnfoncee);
     bouton.setRolloverIcon(imagePassage);
     bouton.setRolloverEnabled(true);
     getContentPane().add(bouton, "Center");

     JPanel panneau = new JPanel();
     panneau.add(bouton);
     setContentPane(panneau);
     setSize(200,100);
     setVisible(true);
   }

   public static void main(String [] args){
      JFrame frame = new swing4();
   }
}
```

4.1. Evénements

Lors de la sélection d'un bouton du groupe, il y a plusieurs événements qui peuvent être émis :

- Un événement de type Action
- Un événement de type Item émis par le bouton sélectionné
- Un événement de type Item émis par le bouton désélectionné s'il y en a un

```
package swing;

import javax.swing.*;
import javax.swing.border.LineBorder;

import java.awt.Color;
import java.awt.event.*;

public class Bouton extends JFrame {

	public static void main(String[] args) {

		// evenement du bouton
		ActionListener eventbutton = new ActionListener() {
			public void actionPerformed(ActionEvent event) {
				System.out.println("Elément de menu [" + event.getActionCommand() + "] utilisé.");
			}
		};
		// créer un frame
		JFrame frame = new JFrame("Exemple JButton");
		// créer un bouton
		JButton btn = new JButton();

		btn.setSize(40, 40);
		btn.setBorder(new LineBorder(Color.BLACK));
		// définir la position du bouton
		btn.setBounds(70, 80, 40, 40);
		btn.addActionListener(eventbutton);
		// rendre le bouton transparent
		btn.setOpaque(false);
		// enlever la zone de contenu
		btn.setContentAreaFilled(false);

		frame.add(btn);
		frame.setSize(250, 250);
		frame.setLayout(null);
		frame.setVisible(true);
	}
}
```

5. TextArea

La classe JTextArea est un composant qui permet la saisie de texte simple en mode multiligne. Le modèle utilisé par ce composant est le PlainDocument : il ne peut donc contenir que du texte brut sans éléments multiples de formatage.

JTexteArea propose plusieurs méthodes pour ajouter du texte dans son modèle :

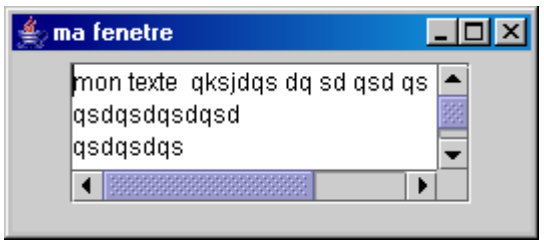
- soit fournir le texte en paramètre du constructeur utilisé
- soit utiliser la méthode setText() qui permet d'initialiser le texte du composant
- soit utiliser la méthode append() qui permet d'ajouter du texte à la fin de celui contenu dans le composant
- soit utiliser la méthode insert() qui permet d'insérer du texte dans le composant à une position données en caractères

```
import javax.swing.*;

public class JTextArea1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();

    JTextArea  textArea1 = new JTextArea ("mon texte");

    pannel.add(textArea1);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```

5.1. JScrollPane

Par défaut, la taille du composant augmente au fur et à mesure de l'augmentation de la taille du texte qu'il contient. Pour éviter cet effet, il faut encapsuler le JTextArea dans un JScrollPane.

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

public class JTextArea1 {

  public static void main(String argv[]) {

    JFrame f = new JFrame("ma fenetre");
    f.setSize(300, 100);
    JPanel pannel = new JPanel();
    JTextArea  textArea1 = new JTextArea ("mon texte");
    JScrollPane scrollPane = new JScrollPane(textArea1);
    scrollPane.setPreferredSize(new Dimension(200,70));
    pannel.add(scrollPane);
    f.getContentPane().add(pannel);
    f.setVisible(true);
  }
}
```



6. Les onglets

La classe javax.swing.JTabbedPane encapsule un ensemble d'onglets. Chaque onglet est constitué d'un titre, d'un composant et éventuellement d'une image.

Pour utiliser ce composant, il faut :

- instancier un objet de type JTabbedPane
- créer le composant de chaque onglet
- ajouter chaque onglet à l'objet JTabbedPane en utilisant la méthode addTab()

```
import java.awt.Dimension;
import java.awt.event.KeyEvent;

import javax.swing.*;

public class TestJTabbedPane1 {

  public static void main(String[] args) {
    JFrame f = new JFrame("Test JTabbedPane");
    f.setSize(320, 150);
    JPanel pannel = new JPanel();

    JTabbedPane onglets = new JTabbedPane(SwingConstants.TOP);

    JPanel onglet1 = new JPanel();
    JLabel titreOnglet1 = new JLabel("Onglet 1");
    onglet1.add(titreOnglet1);
    onglet1.setPreferredSize(new Dimension(300, 80));
    onglets.addTab("onglet1", onglet1);

    JPanel onglet2 = new JPanel();
    JLabel titreOnglet2 = new JLabel("Onglet 2");
    onglet2.add(titreOnglet2);
    onglets.addTab("onglet2", onglet2);

    onglets.setOpaque(true);
    pannel.add(onglets);
    f.getContentPane().add(pannel);
    f.setVisible(true);

  }
}
```

7. Les menus

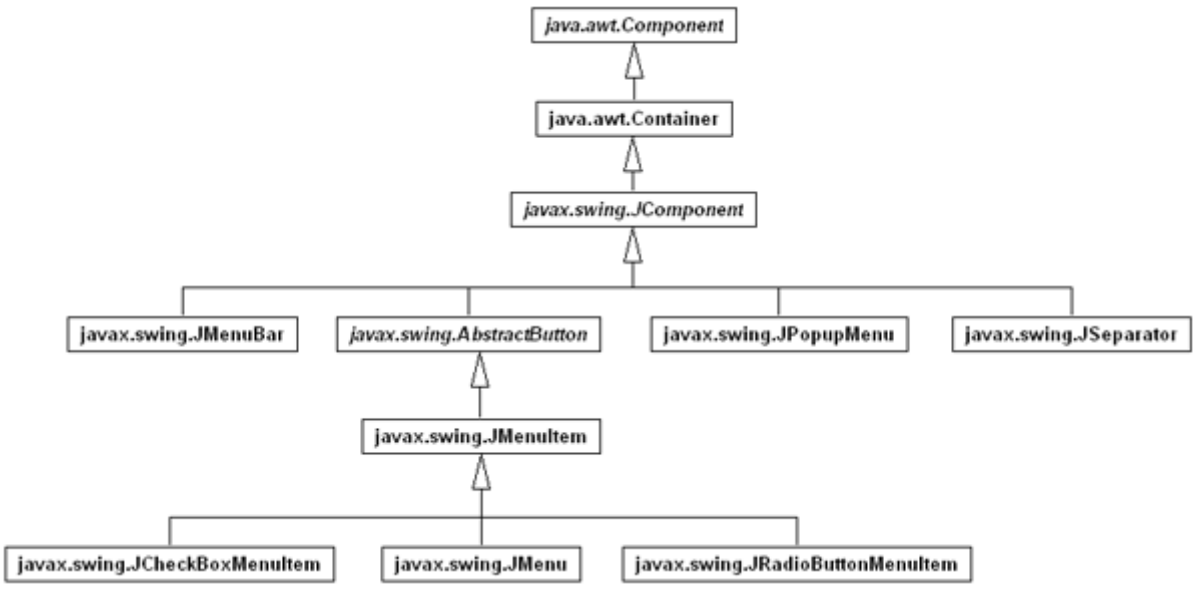
Les menus de Swing proposent certaines caractéristiques intéressantes en plus de celles proposées par un menu standard :

- les éléments de menu peuvent contenir une icône
- les éléments de menu peuvent être de type bouton radio ou case à cocher
- les éléments de menu peuvent avoir des raccourcis clavier (accelerators)

Les éléments de menus cliquables héritent de la classe JAbstractButton.

JMenu hérite de la classe JMenuItem et non pas l'inverse car chaque JMenu contient un JMenuItem implicite qui encapsule le titre du menu.

La plupart des classes utilisées pour les menus implémentent l'interface MenuElement. Cette interface définit des méthodes pour la gestion des actions standards de l'utilisateur. Ces actions sont gérées par la classe MenuSelectionManager.



```java
package swing;

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

public class Menu2 extends JMenuBar {

  public Menu2() {

    // Listener générique qui affiche l'action du menu utilisé
    ActionListener afficherMenuListener = new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        System.out.println("Elément de menu [" + event.getActionCommand()
            + "] utilisé.");
      }
    };

    // Création du menu Fichier
    JMenu fichierMenu = new JMenu("Fichier");
    JMenuItem item = new JMenuItem("Nouveau", 'N');
    item.addActionListener(afficherMenuListener);
    fichierMenu.add(item);
    item = new JMenuItem("Ouvrir", 'O');
    item.addActionListener(afficherMenuListener);
    fichierMenu.add(item);
    item = new JMenuItem("Sauver", 'S');
    item.addActionListener(afficherMenuListener);
    fichierMenu.insertSeparator(1);
    fichierMenu.add(item);
    item = new JMenuItem("Quitter");
    item.addActionListener(afficherMenuListener);
    fichierMenu.add(item);

    // Création du menu Editer
    JMenu editerMenu = new JMenu("Editer");
    item = new JMenuItem("Copier");
    item.addActionListener(afficherMenuListener);

    editerMenu.add(item);
    item = new JMenuItem("Couper");
    item.addActionListener(afficherMenuListener);

    editerMenu.add(item);
    item = new JMenuItem("Coller");
    item.addActionListener(afficherMenuListener);

    editerMenu.add(item);

    // Création du menu Divers
    JMenu diversMenu = new JMenu("Divers");
    JMenu sousMenuDiver1 = new JMenu("Sous menu 1");

    item.addActionListener(afficherMenuListener);
    item = new JMenuItem("Sous menu 1 1");
    sousMenuDiver1.add(item);
    item.addActionListener(afficherMenuListener);
    JMenu sousMenuDivers2 = new JMenu("Sous menu 1 2");
    item = new JMenuItem("Sous menu 1 2 1");
    sousMenuDivers2.add(item);
    sousMenuDiver1.add(sousMenuDivers2);

    diversMenu.add(sousMenuDiver1);
    item = new JCheckBoxMenuItem("Validé");
    diversMenu.add(item);
    item.addActionListener(afficherMenuListener);
    diversMenu.addSeparator();
    ButtonGroup buttonGroup = new ButtonGroup();
    item = new JRadioButtonMenuItem("Cas 1");
    diversMenu.add(item);
    item.addActionListener(afficherMenuListener);
    buttonGroup.add(item);
    item = new JRadioButtonMenuItem("Cas 2");
    diversMenu.add(item);
    item.addActionListener(afficherMenuListener);
    buttonGroup.add(item);
    diversMenu.addSeparator();
    diversMenu.add(item = new JMenuItem("Autre",
        new ImageIcon("about_32.png")));
    item.addActionListener(afficherMenuListener);

    // ajout des menus à la barre de menus
    add(fichierMenu);
    add(editerMenu);
    add(diversMenu);
  }

  public static void main(String s[]) {
        Imagebg app = new Imagebg("Auberge du Poney");


        app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        app.setJMenuBar(new Menu2());
        app.setMinimumSize(new Dimension(250, 200));
        app.pack();
```

```
            app.setVisible(true);
        }
    }
```

8. TP SWING

TP SWING

TP - Algorithme

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: TP - Algorithme

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:22

Table des matières

1. Reverse

Écrivez une méthode qui inverse une chaîne.

Exemple:
reverse('WayToLearnX')

Sortie prévue:
XnraeLoTyaW

Remarque : pour obtenir la valeur d'un objet string à la position utilisez ***`str.charAt(index)`***

```
import java.util.*;

public class Main {

    public static String reverse(String str) {
        // Écrivez votre code ici

        return str;
    }

    public static void main(String[] args) {
        String rev = reverse("WayToLearnX");
        System.out.print(rev);
    }
}
```

1.1. Solution

```java
package algorithme;

public class Tp02_reverse {

	public static void main(String[] args) {
		String rev = reverse("WayToLearnX");
		System.out.print(rev);
	}

	public static String reverse(String str) {
		String res = "";

		for (int i = 0; i < str.length(); i++) {
			res += str.charAt(str.length() - i - 1);
		}
		return res;
	}
}
```

2. Maximum

Écrivez une méthode qui renvoie le plus grand entier dans un tableau.

Vous pouvez supposer que le tableau contient au moins un élément.

Exemple:
int[] tab = {1, 2, 9, 4};

Sortie prévue:
9

2.1. Solution

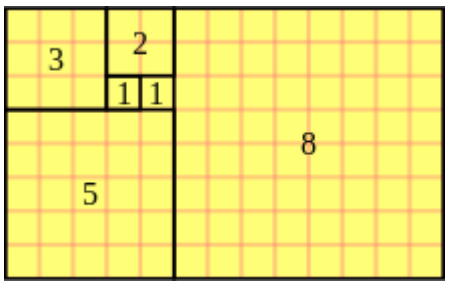
```java
package algorithme;

public class Tp03_maximum {

	public static void main(String[] args) {
		int[] tab = { 1, 2, 9, 4 };
		int max = maximum(tab);
		System.out.print(max);
	}

	public static int maximum(int[] tab) {
		int max = tab[0];
		for (int i : tab) {
			if (i > max) {
				max = i;
			}
		}
		return max;
	}

}
```

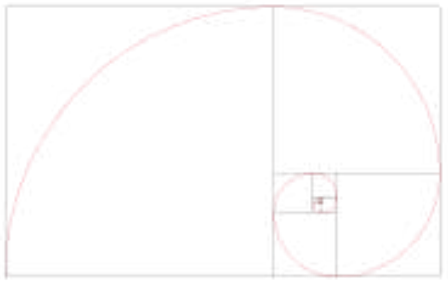
3. Fibonacci

Écrivez une méthode qui renvoie le nième élément d'une séquence de Fibonacci. Une séquence de Fibonacci est une série de nombres: 0, 1, 1, 2, 3, 5, 8, 13, 21, … Le nombre suivant est trouvé en additionnant les deux nombres précédents.

Exemple:
fibonacci(8)

Sortie prévue:
21





3.1. Solution

Récursivité : Attention (plus gourmand, mais code plus léger)

```java
package algorithme;

public class Tp04_fibonacci {

	public static void main(String[] args) {
		int nbr = 8;
		System.out.print(fibonacci(nbr));
	}

	public static int fibonacci(int n) {
		if (n == 1) {
			return 1;
		} else if (n == 0) {
			return 0;
		} else {
			return fibonacci(n - 1) + fibonacci(n - 2);
		}
	}

}
```

3.2. Solution 2

Autre solution sans la récursivité :

```java
package algorithme;

public class Fibonacci_02 {

	public static void main(String[] args) {
		System.out.println(fibonacci(8));
	}

	public static int fibonacci(int nb) {
		int[] result = new int[nb + 1];
		result[0] = 0;
		result[1] = 1;

		for (int i = 2; i <= nb; i++) {

			result[i] = result[i - 1] + result[i - 2];
		}
		return result[nb];
	}

}
```

4. Prix des billets d’autocar

Dans cet exercice, on veut gérer une ligne de car qui relie Vierzon à Orléans en passant par Salbris, Nouans, Lamotte-Beuvron et La Ferté Saint-Aubin.

Chaque tronçon du trajet a un certain prix et le prix d’un billet est la somme des prix des tronçons qu’il comporte. La ligne fonctionne dans les deux sens, le prix est identique dans les deux sens.

On va stocker dans un tableau les noms des villes, en les mettant dans l’ordre dans lequel elles sont desservies (l’ordre donné ci-dessus). Dans un autre tableau, on va stocker les prix des différents tronçons. Ces prix sont les suivants :

départ	arrivée	prix
Vierzon	Salbris	3.20
Salbris	Nouans	1.80
Nouans	Lamotte-Beuvron	2.30
Lamotte-Beuvron	La Ferté Saint-Aubin	4.20
La Ferté Saint-Aubin	Orléans	5.00

Avant de commencer à programmer, dessinez les deux tableaux sur papier et réfléchissez aux algorithmes permettant de répondre aux différentes questions.

Question 1

1. écrivez une méthode permettant de retrouver l’indice d’une ville dont on donne le nom en paramètre (c’est à dire sa position dans le premier tableau).
2. écrivez une méthode qui calcule le prix d’un trajet étant donnés les noms des villes de départ et d’arrivée.
3. Bonus : Utiliser try { } catch (Exception ex) {} pour générer une erreur si "Ville inconnue!"

Question 2

On veut instaurer des tarifs dégressifs selon le nombre de tronçons parcourus : le premier tronçon est payé à plein tarif, le second avec 10% de réduction, le second avec 20%, etc. Ecrivez ou adaptez une méthode qui réalise le calcul du prix d’un trajet selon ce principe.

4.1. Solution 1

```java
package tp01_autocar;

public class Tp01 {

	static double[] prix = { 0.00, 3.20, 1.80, 2.30, 4.20, 5.00 };
	static String[] villes = { "Vierzon", "Salbris", "Nouans", "Lamotte-Beuvron", "La Ferté Saint-Aubin", "Orléans" };
	static int troncon;

	public static void main(String[] args) {

		// Terminal.ecrireIntln(ville("test"));genere une erreur
		System.out.println("Index de la ville : " + ville(villes[0]));
		System.out.println("Prix du trajet aller " + villes[0] + "/" + villes[5] + " : " + trajet(villes[0], villes[5])
				+ " - Tronçon : " + troncon);
		System.out.println("Prix du trajet retour " + villes[5] + "/" + villes[0] + " : " + trajet(villes[5], villes[0])
				+ " - Tronçon : " + troncon);
		System.out.println("Prix du trajet aller " + villes[1] + "/" + villes[4] + " : " + trajet(villes[1], villes[4])
				+ " - Tronçon : " + troncon);
		System.out.println("Prix du trajet retour " + villes[4] + "/" + villes[1] + " : " + trajet(villes[4], villes[1])
				+ " - Tronçon : " + troncon);

	}

	/*
	 * Méthode pour calculer prix traget avec option degressif
	 */
	public static double trajet(String name1, String name2) {

		double prixtrajet = 0;
		boolean ville2 = false;
		troncon = 0;

		try {
			// traget aller
			for (int i = 0; i < villes.length; i++) {
				// trouve la ville1
				if (villes[i] == name1) {
					for (int j = i + 1; j < villes.length; j++) {
						prixtrajet += prix[j];
						troncon += 1;
						// trouve la ville2
						if (villes[j] == name2) {
							ville2 = true;
							break;
						}
					}

					// trajet retour
					if (ville2 == false) {
						prixtrajet = troncon = 0;
						for (int j = i; j >= 0; j--) {
							// trouve la ville2
							if (villes[j] == name2) {
								ville2 = true;
								break;
							}
							prixtrajet += prix[j];
							troncon += 1;
						}

					}
					break;
				}
			}

			if (prixtrajet == 0 || ville2 == false) {
				// ...ou génére une erreur
				villes[villes.length] = "Erreur";
			}

		} catch (Exception ex) {
			System.out.println("Ville inconnue!");
			throw ex;
		}

		return prixtrajet;

	}

	/*
	 * Méthode pour trouver l'index d'une ville
	 */
	public static int ville(String name) {
```

```
                try {
                        for (int i = 0; i < villes.length; i++) {
                                // trouve la ville
                                if (villes[i] == name) {

                                        return i;
                                }
                        }

                        // ...ou génére une erreur
                        villes[villes.length] = "Erreur";

                } catch (Exception ex) {
                        System.out.println("Ville inconnue!");
                        throw ex;
                }

                return villes.length;

        }

}
```

4.2. Solution 1 bis

```java
package algorithme;

import java.util.stream.IntStream;

public class TP_Autocar_test {

	static String[] tabVilles = { "Vierzon", "Salbris", "Nouans", "Lamotte-Beuvron", "La Ferté Saint-Aubin",
			"Orléans" };
	static double[] tabPrix = { 3.20, 1.80, 2.30, 4.20, 5.00 };

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		///objet
		//TP_Autocar_test test = new TP_Autocar_test();
		//System.out.println(test.indiceVille("Lamotte-Beuvron"));
		System.out.println(indiceVille("Lamotte-Beuvron"));
		System.out.println(CalculTrajet(tabVilles[0], tabVilles[5]));

	}

	public static int indiceVille(String ville) {

		int indice = -1;

		for (int i = 0; i < tabVilles.length; i++) {
			if (ville == tabVilles[i]) {
				indice = i;
			}
		}

		return indice;

	}

	public static float CalculTrajet(String Depart, String Arrivee) {
		int IndexDepart = indiceVille(Depart);
		int IndexArrivee = indiceVille(Arrivee);
		int indexMin = Integer.min(IndexDepart, IndexArrivee);
		int indexMax = Integer.max(IndexDepart, IndexArrivee);

		return (float) IntStream.range(indexMin, indexMax).mapToDouble(i -> tabPrix[i]).sum();
	}
}
```

4.3. Solution 2

```java
package tp01_autocar;

public class Tp01 {

	static double[] prix = { 0.00, 3.20, 1.80, 2.30, 4.20, 5.00 };
	static String[] villes = { "Vierzon", "Salbris", "Nouans", "Lamotte-Beuvron", "La Ferté Saint-Aubin", "Orléans" };
	static double troncon;

	public static void main(String[] args) {

		System.out.println("Prix du trajet aller normal " + villes[1] + "/" + villes[4] + " : "
				+ trajet(villes[1], villes[4], false));
		System.out.println("Prix du trajet aller degressif " + villes[1] + "/" + villes[4] + " : "
				+ trajet(villes[1], villes[4], true));

		System.out.println("Prix du trajet retour normal " + villes[5] + "/" + villes[0] + " : "
				+ trajet(villes[5], villes[0], false));
		System.out.println("Prix du trajet retour degressif " + villes[5] + "/" + villes[0] + " : "
				+ trajet(villes[5], villes[0], true));
	}

	/*
	 * Méthode pour calculer prix traget avec option degressif
	 */
	public static double trajet(String name1, String name2, boolean degressif) {

		double prixtrajet = 0.00;
		double remise = 0.00;
		boolean ville2 = false;
		troncon = 0;

		try {
			// traget aller
			for (int i = 0; i < villes.length; i++) {
				// trouve la ville1
				if (villes[i] == name1) {
					for (int j = i + 1; j < villes.length; j++) {
						troncon += 1;
						if (degressif && troncon > 1) {
							remise = prix[j] * ((troncon - 1) / 10);
						}
						prixtrajet += prix[j] - remise;

						// trouve la ville2
						if (villes[j] == name2) {
							ville2 = true;
							break;
						}
					}

					// trajet retour
					if (ville2 == false) {
						prixtrajet = troncon = 0;
						for (int j = i; j >= 0; j--) {
							// trouve la ville2
							if (villes[j] == name2) {
								ville2 = true;
								break;
							}
							troncon += 1;
							if (degressif && troncon > 1) {
								remise = prix[j] * ((troncon - 1) / 10);
							}
							prixtrajet += prix[j] - remise;

						}
					}
					break;
				}
			}

			if (prixtrajet == 0 || ville2 == false) {
				// ...ou génére une erreur
				villes[villes.length] = "Erreur";
			}

		} catch (Exception ex) {
			System.out.println("Ville inconnue!");
			throw ex;
		}

		return prixtrajet;
```

```
}

/*
 * Méthode pour trouver l'index d'une ville
 */
public static int ville(String name) {

        try {
                for (int i = 0; i < villes.length; i++) {
                        // trouve la ville
                        if (villes[i] == name) {

                                return i;
                        }
                }

                // ...ou génére une erreur
                villes[villes.length] = "Erreur";

        } catch (Exception ex) {
                System.out.println("Ville inconnue!");
                throw ex;
        }

        return villes.length;

}
```

TP - Classes & Constructeurs

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: TP - Classes & Constructeurs

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:22

Table des matières

1. Compte TP1

Voici le texte d'une classe représentant de façon sommaire un compte bancaire et les opérations bancaires courantes :

```
public class Compte{
        int solde = 0;

        void deposer(int montant) {
                solde = solde + montant;
        }

        void retirer(int montant) {
                solde = solde - montant;
        }

        void virerVers(int montant, Compte destination) {
                this.retirer(montant);
                destination.deposer(montant);
        }

        void afficher() {
                System.out.println("solde: " + solde);

        }

}
```

Créez deux comptes que vous affecterez à deux variables. Ecrivez le code correspondant aux opérations suivantes :

- dépot de 500 euros sur le premier compte.
- dépot de 1000 euros sur le second compte.
- retrait de 10 euros sur le second compte.
- virement de 75 euros du premier compte vers le second.
- affichage des soldes des deux comptes.

Vous mettrez le code java correspondant à cette question dans la méthode main d'une nouvelle classe appelée TesteCompte. Vous compilerez et testerez ce programme.

1.1. Solution

```java
package tp02_compte;

public class TesteCompte {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Compte compte1 = new Compte();
		Compte compte2 = new Compte();

		compte1.deposer(500);
		compte2.deposer(1000);
		compte2.retirer(10);
		compte1.virerVers(75, compte2);
		compte1.afficher();
		compte2.afficher();

	}

}
```

```
solde: 425
solde: 1065
```

2. Compte TP2

Créez un tableau de dix comptes.

Pour cela, notez bien qu'il faut d'abord créer le tableau puis créer successivement les dix comptes à mettre dans les dix cases de ce tableau.

Dans chaque case, faites un dépot de 200 euros plus une somme égale à 100 fois l'indice du compte dans le tableau.

Ensuite, vous ferez un virement de 20 euros de chaque compte vers chacun des comptes qui le suivent dans le tableau (par exemple, du compte d'indice 5, il faut faire des virements vers les comptes d'indice 6, 7, 8 et 9).

Enfin, vous afficherez les soldes de tous les comptes. Ici encore, vous testerez et compilerez le code proposé.

2.1. Solution

```java
package tp02_compte;

public class TesteCompte {

	public static void main(String[] args) {

		//tableau de type compte
		Compte[] tab = new Compte[10];

		for (int i = 0; i < tab.length; i++) {
			//créer successivement les dix comptes
			tab[i] = new Compte();
			// faites un dépot de 200 euros plus une somme égale à 100 fois l'indice du compte dans le tableau.
			tab[i].deposer(200 + (100*i));
		}

		for (int i = 0; i < tab.length; i++) {

			//virement de 20 euros de chaque compte vers chacun des comptes qui le suivent
			for (int j = i; j < tab.length; j++) {
				tab[i].virerVers(20, tab[j]);
			}

		}

		//soldes de tous les comptes
		for (int i = 0; i < tab.length; i++) {

			System.out.println("Compte : " + i);
			tab[i].afficher();

		}

		Compte compteTab = new Compte();


		Compte compte1 = new Compte();
		Compte compte2 = new Compte();

		compte1.deposer(500);
		compte2.deposer(1000);
		compte2.retirer(10);
		compte1.virerVers(75, compte2);
		System.out.println("Compte test 1 : ");
		compte1.afficher();
		System.out.println("Compte test 2 : ");
		compte2.afficher();

	}

}
```

3. Compte TP3

Complétez la classe Compte avec une information supplémentaire : le nom du titulaire du compte (type String).

Vous modifierez la méthode d’affichage pour qu’elle affiche cette information.

Créez un constructeur pour la classe Compte. Ce constructeur doit prendre en paramètre le nom du titulaire du compte.

Donnez le code de création d’un compte qui appelle ce constructeur.

Faut-il prévoir des méthodes permettant de changer le nom du titulaire du compte ?

3.1. Solution

```java
package tp02_compte;

public class Compte {
	int solde = 0;
	String titulaire;
	int numero;

	//constructeur par defaut
	public Compte() {

	}

	//constructeur avec titulaire
	public Compte(String titulaire) {
		this.titulaire = titulaire;
	}

	void afficher() {
		System.out.println("solde: " + solde + " nom:" + titulaire);
	}

	void deposer(int montant) {
		solde = solde + montant;
	}

	void retirer(int montant) {
		solde = solde - montant;
	}

	void virerVers(int montant, Compte destination) {
		this.retirer(montant);
		destination.deposer(montant);
	}

	//changer nom du compte
	void changeTitulaire(String titulaire) {
		this.titulaire = titulaire;

	}

}
```

```java
Compte compte3 = new Compte("PAP JAVA");
compte3.deposer(500);
System.out.println("Compte test 3 : ");
compte3.afficher();
compte3.changeTitulaire("JAVA TP02");
compte3.afficher();
```

TP - Classes, surcharges et List Object

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: TP - Classes, surcharges et List Object

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:22

Table des matières

1. ArrayList

Écrivez un programme Java pour créer un ArrayList nommé « languages », ajoutez des chaîne(Ex: PHP, Java, C++, Python) , vous devez dans cet ordre :

1. Affichez la collection.
2. Parcourir tous les éléments d'un ArrayList, en utilisant la boucle for.
3. Insérer l'élément « Pascal » en troisième position, en specifiant l'index d'insertion dans l'ArrayList. Utiliser `add`.
4. Récupérer le troisième élément à partir d'un ArrayList et l'afficher. Utiliser `get`.
5. Mettre à jour le troisième élément de l'ArrayList par « COBOL » et afficher la collection. Utiliser `set`.
6. Supprimer le troisième élément d'un ArrayList et afficher la collection. Utiliser `remove`.
7. Rechercher si un élément est présent dans un ArrayList, exemple "Java". Utiliser `contains`.

Sortie prévue:
[PHP, Java, C++, Python]

1.1. Solution

```java
package arraylist;

import java.util.ArrayList;
import java.util.List;

public class TP_arraylist {
        public static void main(String[] args) {

                List<String> languages = new ArrayList<String>();

                languages.add("PHP");
                languages.add("Java");
                languages.add("C++");
                languages.add("Python");


                //1. Affichez la collection
                System.out.println(languages);

                //2. Parcourir tous les éléments d’un ArrayList, en utilisant la boucle for.
                for (String lang : languages) {
                        System.out.println(lang);
                }

                // 3. Insérer l'élément en troisième position
                languages.add(2, "Pascal");

                // 4. Récupérer le troisième élément
                String lang = languages.get(2);
                System.out.println("Le troisième élément: " + lang);

                // 5. Mettre à jour le troisième élément avec "COBOL"
                languages.set(2, "COBOL");
                System.out.println(languages);

                // 6. Supprimer le troisième élément
                languages.remove(2);
                System.out.println("Aprés la suppression : " + languages);

                // 7. Trouver la valeur Java
                if (languages.contains("Java")) {
                        System.out.println("L'élément existe dans la liste");
                } else {
                        System.out.println("L'élément n'existe pas dans la liste");
                }
        }
}
```

2. Reverse

Corrigez la classe suivante afin qu'elle soit compilée et que la méthode reverse renvoie un ArrayList contenant des entiers dans l'ordre inverse de la liste des paramètres ArrayList.

Exemple:
[1, 2, 3, 4, 5, 6, 7]

Sortie prévue:
[7, 6, 5, 4, 3, 2, 1]

```
public static ArrayList reverse(ArrayList<Integer> liste)
    {
        ArrayList<int> res = new ArrayList<int>();
        for (Integer i: liste)
        {

        }
        return res;
    }

    public static void main(String[] args)
    {
        ArrayList<Integer> liste = new ArrayList<Integer>();

        int[] nbrs = {1, 2, 3, 4, 5, 6, 7};

        for (int i = 0; i < nbrs.length; i++)
        {
            liste.add(nbrs[i]);
        }

        ArrayList<Integer> res = reverse(liste);
        System.out.println(res);
    }
```

2.1. Solution

Remplacez dans la methode `reverse`, int par Integer car ArrayLists stocke uniquement des objets et int est un type primitive. Ajoutez `res.add(0, i);` de sorte que chaque élément de la liste, le paramètre, soit ajouté devant l'élément précédemment ajouté.

```
public static ArrayList<Integer> reverse(ArrayList<Integer> liste)
   {
       ArrayList<Integer> res = new ArrayList<Integer>();
       for (Integer i: liste)
       {
               res.add(0, i);
       }
       return res;
   }
   public static void main(String[] args)
   {
       ArrayList<Integer> liste = new ArrayList<Integer>();

       int[] nbrs = {1, 2, 3, 4, 5, 6, 7};

       for (int i = 0; i < nbrs.length; i++)
       {
           liste.add(nbrs[i]);
       }

       ArrayList<Integer> res = reverse(liste);
       System.out.println(res);
   }
```

3. Cocktail

Cet exercice a pour but de réfléchir sur la conception d'un programme, sa structuration en classes.

Il s'agit de réaliser ce programme et de concevoir son architecture.

On fait des cocktails avec différents liquides (alcools, sodas, jus de fruits).

Il y a des recettes de cocktails qui indiquent seulement les proportions en cl.

Ces recettes peuvent s'appliquer à des quantités plus ou moins grandes selon les besoins du moment.

- Question : quelles classes faut-il créer ?
- Quelles informations faut-il dans chaque classe ?
- Quelles méthodes faut-il écrire, et dans quelle classe les mettre ?

Pour chaque méthode, précisez le type des paramètres et de la valeur de retour.

Sachant que pour réaliser la recette du Mojito vous devez ajouter les ingrédients suivant :

- Rhum Cubain 6 cl
- Citron verts 3 cl
- Sirop de sucre de canne 2 cl
- Eau Gazeuse 38 cl

Créez les Class Cocktail, Ingrédient et TestCocktail.
Ecrivez la méthode permettant d'obtenir le résultat suivant en utilisant `System.out.println(mojito);` dans la class TestCocktail.

```
Mojito[rhum cubain 6 cl, citrons verts 3 cl, sucre de canne 2 cl, eau gazeuse 38 cl]
```

3.1. Solution - Questions

Question : quelles classes faut-il créer?

Les class Cocktail, Ingredient et TesteCocktail

Quelles informations faut-il dans chaque classe?

Pour la class Cocktail :

- un nom de type string
- une Liste de type Ingredient

Pour la class Ingredient :

- un nom de type string
- une quantité en cl

Quelles constructeur et méthodes faut-il écrire, et dans quelle classe les mettre?

Pour la class Cocktail :

- Un constructeur avec le nom et l'initialisation de ingredients
- la methode addIngredient(Ingredient ing)

Pour la class Ingredient :

- Un constructeur avec le nom et la quantité en cl
- Une surcharge de la methode toString() pour afficher le nom et la quantité

3.2. Solution - TesteCocktail

```java
package tp03_cocktail;

public class TesteCocktail {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Cocktail mojito = new Cocktail("Mojito");

		mojito.addIngredient(new Ingredient("rhum cubain", 6));
		mojito.addIngredient(new Ingredient("citrons verts", 3));
		mojito.addIngredient(new Ingredient("sucre de canne", 2));
		mojito.addIngredient(new Ingredient("eau gazeuse", 38));

		System.out.println(mojito);

	}

}
```

3.3. Solution - Cocktail

```java
package tp03_cocktail;

import java.util.ArrayList;
import java.util.List;

public class Cocktail {

	private String name;

	private List<Ingredient> ingredients;

	public Cocktail(String name) {
		this.name = name;
		this.ingredients = new ArrayList<Ingredient>();
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return this.name;
	}

	public void addIngredient(Ingredient ing) {
		this.ingredients.add(ing);
	}

	public Ingredient getIngredient(int index) {
		return this.ingredients.get(index);
	}

	// methode surcharge Override
	public String toString() {
		String s = this.name + "[";
		for (int i = 0; i < this.ingredients.size(); i++) {
			if (i > 0)
				s += ", ";
			s += this.ingredients.get(i);
		}
		s += "]";
		return s;
	}

}
```

3.4. Solution - Ingredient

```java
package tp03_cocktail;

public class Ingredient {
	private String name;
	//quantité en cl
	private int quantity;

	public Ingredient(String name, int quantity) {
		this.name = name;
		this.quantity  =  quantity;
	}
	public String getName() {
		return this.name;
	}
	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}
	public int getQuantity() {
		return this.quantity;
	}

	//Surcharge
	public String toString() {
		return this.name+" "+this.quantity+" cl";
	}

}
```

TP - Héritage

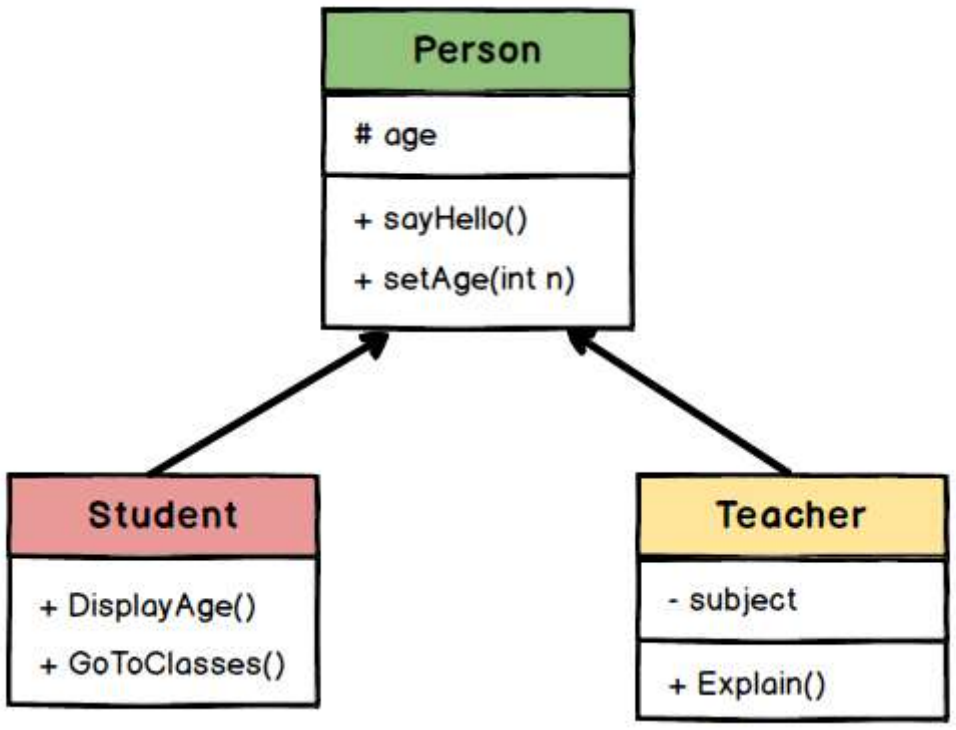
Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: TP - Héritage

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:22

Table des matières

1. TP - Etudiant

– Créez une classe « Person »
– Créez une classe « Student » et une autre classe « Teacher », les deux héritent de la classe « Person ».
– La classe « Student » aura une méthode publique « GoToClasses », qui affichera à l'écran « I'm going to class. ».
– La classe « Teacher » aura une méthode publique « Explain », qui affichera à l'écran « Explanation begins ». En plus, il aura un attribut privé « subject » de type string.
– La classe « Person » doit avoir une méthode « SetAge(int n) » qui indiquera la valeur de leur âge (par exemple, 15 years old).
– La classe « Student » aura une méthode publique « DisplayAge » qui écrira sur l'écran « My age is: XX years old ».
– Vous devez créer une autre classe de test appelée « Test » qui contiendra « Main » et:
– Créez un objet Person et faites-lui dire « Hello »
– Créer un objet Student, définir son âge à 15 ans, faites-lui dire « Hello », « I'm going to class. » et afficher son âge
– Créez un objet Teacher, 40 ans, demandez-lui de dire « Hello » puis commence l'explication.



1.1. Solution

```java
class Person
{
    protected int age;

    public void sayHello()
    {
    System.out.println("Hello");
    }

    public void setAge(int n)
    {
        age = n;
    }
}
//La classe Student
class Student extends Person
{
    public void goToClasses()
    {
    System.out.println("I’m going to class.");
    }
    public void displayAge()
    {
    System.out.println("My age is: "+ age +" years old");
    }
}
//La classe Teacher
class Teacher extends Person
{
    private String subject;

    public void explain()
    {
    System.out.println("Explanation begins");
    }
}
//Classe de test
public class Main
{
  public static void main(String[] args)
  {
            Person p = new Person();
            p.sayHello();

            Student s = new Student();
            s.goToClasses();
            s.setAge(15);
            s.sayHello();
            s.displayAge();

            Teacher t = new Teacher();
            t.setAge(40);
            t.sayHello();
            t.explain();
  }
}
```

2. TP - Appartement

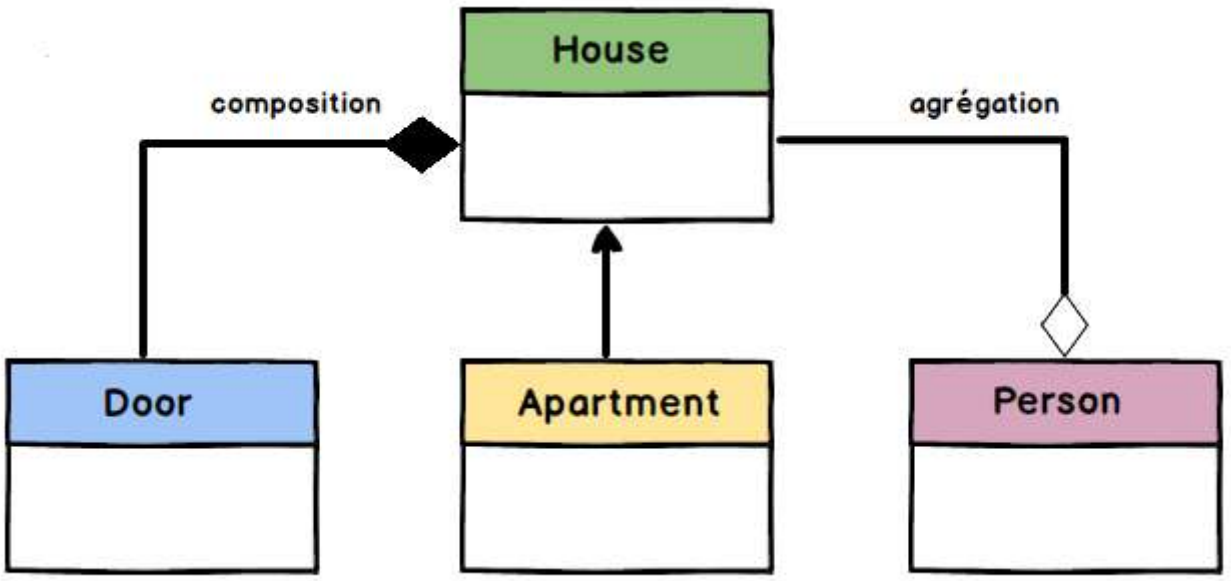
Créez une classe « House », avec un attribut « surface », un constructeur qui définit sa valeur et une méthode « Display » pour afficher « Je suis une maison, ma surface est de XXX m2 » (XXX: la valeur de surface). Incluez aussi des getters et des setters pour la surface.

La classe « House » contiendra une porte (Door). Chaque porte aura un attribut « color » (de type String), et une méthode « Display » qui affichera « Je suis une porte, ma couleur est bleu » (ou quelle que soit la couleur). Inclure un getter et un setter. Créez également la méthode « GetDoor » dans la classe « House ».

La classe « Apartment » est une sous-classe de la classe « House », avec une surface prédéfinie de 50m2.

Créez également une classe Person, avec un nom (de type String). Chaque personne aura une maison. La méthode « Display » pour une personne affichera son nom, les données de sa maison et les données de la porte de cette maison.

Écrivez un Main pour créer un Apartment, une personne pour y vivre et pour afficher les données de la personne.



Notions UML à savoir :

- **La composition** peut être considérer comme une relation “**fait partie de**”, c’est à dire que si un objet Y fait partie d’un objet X alors Y ne peut pas exister sans X. Ainsi si X disparaît alors Y également.
- **L’agrégation** peut être considérer comme une relation de type “**a un**”, c’est à dire que si un objet X a un objet Y alors Y peut vivre sans X.

2.1. Solution

```java
class House
{
  protected int surface;
  protected Door door;

  public House(int surface)
  {
    this.surface = surface;
    door = new Door();
  }

  public int getSurface()
  {
    return surface;
  }

  public void setSurface(int value)
  {
    surface = value;
  }

  public Door getDoor()
  {
    return door;
  }

  public void setDoor(Door d)
  {
    door = d;
  }

  public void display()
  {
    System.out.println("Je suis une maison, ma surface est de "+ surface +" m2.");
  }
}

class Door
{
  protected String color;

  public Door()
  {
    color = "blue";
  }

  public Door(String color)
  {
    this.color = color;
  }

  public String getColor()
  {
    return color;
  }

  public void setColor(String value)
  {
    color = value;
  }

  public void display()
  {
    System.out.println("Je suis une porte, ma couleur est "+ color +".");
  }
}

class Apartment extends House
{
  public Apartment()
  {
    super(50);
  }

  public void display()
  {
    System.out.println("Je suis un appartement, ma surface est " + surface + " m2");
  }
}

class Person
{
  protected String name;
  protected House house;

  public Person()
  {
    name = "Thomas";
    house = new House(150);
  }

  public Person(String name, House house)
  {
    this.name = name;
    this.house = house;
```

```
  }

  public String getName()
  {
    return name;
  }

  public void setName(String value)
  {
    name = value;
  }

  public House getHouse()
  {
    return house;
  }

  public void setHouse(House h)
  {
    house = h;
  }

  public void display()
  {
    System.out.println("Je m'appele "+ name +".");
    house.display();
    house.door.display();
  }
}

//Classe de test
public class Main
{
  public static void main(String[] args)
  {
       Apartment MyApartament = new Apartment();
       Person person = new Person();
       person.name = "Thomas";
       person.house = MyApartament;
       person.display();
  }
}
```

3. TP - Cocktail

Suite du TP Cocktail...

Sachant que vous avez les bouteilles avec les contenants suivants :

- Rhum - Deux bouteilles de 75 cl
- Citron verts - Deux bouteilles d'1 l
- Sucre de canne - Deux bouteilles de 50 cl
- Eau Gazeuse - Dix bouteilles d'1 l

Vous ne pouvez pas modifier les Classes précédentes (Ingrédient et Cocktail).

1. Réaliser la `class Stock` et afficher le contenu du stock en utilisant l'héritage.
2. Ecrivez la méthode dans `Stock` permettant de déterminer combien de Mojito vous pouvez réaliser. Quelle bouteille devez vous achetez en priorité pour faire plus de Mojito ?
3. En fonction de la solution trouvé, compléter votre code pour ajouter le stock manquant.

3.1. TesteCocktail

Rajouter du Rhum :

```java
package heritage.tp01_cocktail;

public class TesteCocktail {

	public static void main(String[] args) {

		// Ajouter les stocks
		Stock rhum = new Stock("rhum cubain", 150);
		new Stock("citrons verts", 200);
		new Stock("sucre de canne", 100);
		new Stock("eau gazeuse", 1000);

		//Afficher stock
		System.out.println(Stock.afficherStock());

		Cocktail mojito = new Cocktail("Mojito");

		mojito.addIngredient(new Ingredient("rhum cubain", 6));
		mojito.addIngredient(new Ingredient("citrons verts", 3));
		mojito.addIngredient(new Ingredient("sucre de canne", 2));
		mojito.addIngredient(new Ingredient("eau gazeuse", 38));

		System.out.println(mojito);
		//Vous pouvez réaliser 25 Mojito Il vous manque une bouteille de rhum cubain
		System.out.println(Stock.nombreCocktail(mojito));

		//Ajout Rhum
		rhum.setQuantity(300);
		//Afficher stock
		System.out.println(Stock.afficherStock());
		System.out.println(Stock.nombreCocktail(mojito));
	}

}
```

3.2. Ingredient

```java
package cocktail;

import java.util.ArrayList;
import java.util.List;

public class Stock extends Ingredient {

	private static List<Stock> stocks = new ArrayList<Stock>();


	public Stock(String name, int quantity) {
		super(name, quantity);
		Stock.stocks.add(this);
	}

	public static String afficherStock() {
		String s = "";
		for (int i = 0; i < Stock.stocks.size(); i++) {
			s += "[" + Stock.stocks.get(i).getName() + ", " + Stock.stocks.get(i).getQuantity() + " cl],";
		}
		return s;
	}

	public void addStock(Stock quantity) {
		Stock.stocks.set(0, quantity);
	}

	public static String nombreCocktail(Cocktail cocktail) {
		int nb = 0;
		int nbmini = 0;
		String message = "";
		for (int i = 0; i < cocktail.ingredients.size(); i++) {
			for (int j = 0; j < Stock.stocks.size(); j++) {

				if (Stock.stocks.get(j).getName() == cocktail.ingredients.get(i).getName()) {
					nb = Stock.stocks.get(j).getQuantity() / cocktail.ingredients.get(i).getQuantity();
					if (i == 0 || nb < nbmini) {
						nbmini = nb;
						message = "Vous pouvez réaliser " + nbmini + " " + cocktail.getName()
								+ " Il vous manque une bouteille de " + Stock.stocks.get(j).getName();
					}
				}

			}

		}
		return message;
	}

}
```

TP - Interface et class abstraite

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: TP - Interface et class abstraite

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:24

Description

On veut écrire un programme pour un jeu vidéo sur le thème du Seigneur des Anneaux de Tolkien. Centraliser vos classes dans le package heroicfantasy.

Table des matières

1. Exercice 1

Un programmeur a déjà écrit la classe Personnage qui suit, et il est impossible de la modifier.

```java
package heroicfantasy;

public abstract class Personnage {

	private String nom;
	private int pointsVie, x, y, vitesse, sous;

	/*
	 * Résumé du rôle de la méthode.
	 *
	 * @param n Nom du personnage
	 * @param x position
	 * @param y position
	 *
	 */
	public Personnage(String n, int x, int y, int v) {
		this.nom = n;
		this.x = x;
		this.y = y;
		this.pointsVie = 100;
		this.vitesse = v;
		this.sous = 0;
	}

	public int getSous() {
		return this.sous;
	}

	public String getNom() {
		return this.nom;
	}

	public int getVitesse() {
		return this.vitesse;
	}

	public void setSous(int s) {
		this.sous = s;
	}

	public int getPointsVie() {
		return this.pointsVie;
	}

	public void setPointsVie(int pv) {
		this.pointsVie = pv;
	}

	/** Le personnage se deplace dans la direction (dx,dy) durant un temps t. */
	public void seDeplacer(int dx, int dy, int t) {
		this.x = (int) (this.x + dx * this.vitesse * t / Math.sqrt(dx * dx + dy * dy));
		this.y = (int) (this.y + dy * this.vitesse * t / Math.sqrt(dx * dx + dy * dy));
	}

	public abstract String parler();

}
```

Il faut représenter dans le programme différents types de personnages :

- les humains qui se déplacent à 5km/h et parlent en disant "Bonjour",
- les elfes qui se déplacent à 7km/h et parlent en disant "Eldarie"
- les nains qui se déplacent à 2km/h et parlent en disant "Groumpf"

1. **Écrire du code permettant de représenter les humains, elfes et nains.**

On veut également représenter les compétences des personnages :

- les guerriers sont des personnages qui peuvent attaquer un autre personnage et possèdent une certaine force.
- les magiciens sont des personnages qui peuvent lancer un sort sur un autre personnage et possèdent un certain niveau de magie.
- les voleurs sont des personnages qui peuvent voler les sous d'un autre personnage et possèdent une certaine dextérité.

Un personnage donné peut avoir plusieurs compétences (par exemple être à la fois guerrier et magicien).

2. **Écrire du code qui permettent de représenter ces compétences dans le programme.**
3. **Écrire du code permettant de représenter les personnages nains qui sont des guerriers.**

Un nain guerrier attaque un autre personnage en lui retirant un nombre de points de vie égal à sa propre force.

4. **Écrire du code permettant de représenter les personnages nains qui sont à la fois guerriers et voleurs.**

Un nain voleur vole un autre personnage en lui retirant un nombre de sous égal à sa propre dextérité (ou moins si le personnage volé possède un nombre de sous inférieur à la dextérité du nain).

Dans une class `testeur` Vous devez afficher le résultat suivant :

```
Bonjour! Je suis Gandalf. Ma vitesse est de 5 km/h. Humain
Bonjour! Je suis Aragorn. Ma vitesse est de 5 km/h. Humain
Eldarie! Je suis Legolas. Ma vitesse est de 7 km/h. Elfe
Groumpf! Je suis Gimli. Ma vitesse est de 2 km/h. Nain guerrier de force 5
Groumpf! Je suis Fili. Ma vitesse est de 2 km/h. Nain guerrier de force 3 et voleur de dexterite 2
```

Exemple de restitution :

```
System.out.print(heros.parler() + "! Je suis " + heros.getNom());
System.out.print(". Ma vitesse est de " + heros.getVitesse() + " km/h. " + heros);
System.out.println(". Je suis " + (heros.getVisible() ? "visible" : "invisible"));
```

1.1. Class Elfe, Humain, Nain

```java
package heroicfantasy;

public class Elfe extends Personnage {
	public Elfe(String nom, int x, int y) {
		super(nom, x, y, 7);
	}

	public String parler() {
		return "Eldarie";
	}

	public String toString() {

		return "Elfe";

	}
}

public class Humain extends Personnage {

	public Humain(String nom, int x, int y) {
		super(nom, x, y, 5);
	}

	public String parler() {
		return "Bonjour";
	}

	public String toString() {

		return "Humain";

	}

}

public class Nain extends Personnage {

	public Nain(String nom, int x, int y) {
		super(nom, x, y, 2);
	}

	public String parler() {
		return "Groumpf";
	}

	// methode surcharge Override
	public String toString() {

		return "Nain";

	}

}
```

1.2. Class Guerrier, Magicien, Voleur

```
public interface Guerrier {

        public void attaque(Personnage p);

        public int getForce();
}

public interface Magicien {

        public void lancerSort(Personnage p);

        public int getMagie();
}

public interface Voleur {

        public void voler(Personnage p);

        public int getDexterite();
}
```

1.3. Class NainGuerrier, NainGuerrierVoleur

```java
public class NainGuerrier extends Nain implements Guerrier {
	private int force;

	public NainGuerrier(String nom, int x, int y, int force) {
		super(nom, x, y);
		this.force = force;
	}

	public void attaque(Personnage p) {
		p.setPointsVie(p.getPointsVie() - this.force);
	}

	public int getForce() {
		return this.force;
	}

	// methode surcharge Override
	public String toString() {

		return super.toString() + " guerrier de force " + this.force;

	}
}

public class NainGuerrierVoleur extends NainGuerrier implements Voleur {
	private int dexterite;

	public NainGuerrierVoleur(String nom, int x, int y, int force, int dexterite) {
		super(nom, x, y, force);
		this.dexterite = dexterite;
	}

	public void voler(Personnage p) {
		if (p.getSous() >= this.dexterite) {
			p.setSous(p.getSous() - this.dexterite);
			this.setSous(this.getSous() + this.dexterite);
		} else {
			this.setSous(this.getSous() + p.getSous());
			p.setSous(0);
		}
	}

	public int getDexterite() {
		return this.dexterite;
	}

	public String toString() {

		return super.toString() + " et voleur de dexterité " + this.dexterite;

	}
}
```

1.4. Class Testeur

```java
package heroicfantasy;

import java.util.ArrayList;
import java.util.List;

public class Testeur {

	public static void main(String[] args) {

		List<Personnage> personnages = new ArrayList<Personnage>();

		Personnage gandalf = new Humain("Gandalf",1,1);
		Personnage aragorn = new Humain("Aragorn",1,1);
		Personnage legolas = new Elfe("Legolas",1,1);
		Personnage gimli = new NainGuerrier("Gimli",1,1,5);
		Personnage fili = new NainGuerrierVoleur("Fili",1,1,3,2);

		personnages.add(gandalf);
		personnages.add(aragorn);
		personnages.add(legolas);
		personnages.add(gimli);
		personnages.add(fili);

		for (Personnage heros : personnages) {
			System.out.print(heros.parler() + "! Je suis " + heros.getNom());
			System.out.println(". Ma vitesse est de " + heros.getVitesse() + " km/h. " + heros);
		}

	}
}
```

2. Exercice 2

On veut représenter les anneaux magiques que peuvent porter les personnages, mais avec comme contrainte qu'**un anneau ne peut exister qu'en lien avec son porteur**, qui est un personnage (un anneau a toujours le même porteur tout au long du jeu).

Il existe plusieurs types d'anneaux, qui ont des effets différents lorsqu'on les passe au doigt, mais on veut imposer que chaque anneau ait un nom et puisse être passer au doigt de son porteur, ou enlevé du doigt de son porteur.

1. Modifier la classe Personnage permettant de représenter les anneaux.
2. Écrivez du code qui permet de représenter l'anneau de Sauron. Cet anneau est d'un type particulier qui permet à son porteur de devenir invisible quand il le passe au doigt, et de redevenir visible quand il l'enlève.
3. Écrivez du code permettant de représenter les hobbits, qui sont des personnages particuliers. Les hobbits parlent en disant "Belle journée ma foi" et se déplacent à 5km/h. Ecrire une méthode main qui crée un hobbit appelé Frodon, situé aux coordonnées (200,100) et un anneau de Sauron dont Frodon est le porteur.
4. L'anneau de Sauron est unique (c'est le seul de son type). Écrivez du code permettant de rendre l'anneau de Sauron unique dans le jeu. **Attention** : il n'est pas possible de définir d'attribut ou de méthode static dans une classe interne.
5. Afficher le résultat ci-dessous en faisant en sorte que `Frodon metAudoigt()` l'anneau :

Belle journée ma foi! Je suis Frodon. Ma vitesse est de 5 km/h. Hobbit. Je suis invisible.

Exemple de restitution sur Testeur :

```
for (Personnage heros : personnages) {
        System.out.print(heros.parler() + "! Je suis " + heros.getNom());
        System.out.print(". Ma vitesse est de " + heros.getVitesse() + " km/h. " + heros);
        System.out.println(". Je suis " + (heros.getVisible() ? "visible" : "invisible"));
}
```

2.1. Class Testeur

```java
package heroicfantasy;

import java.util.ArrayList;
import java.util.List;

import heroicfantasy.Personnage.Anneau;


public class Testeur {

	public static void main(String[] args) {

		List<Personnage> personnages = new ArrayList<Personnage>();

		Personnage gandalf = new Humain("Gandalf",1,1);
		Personnage aragorn = new Humain("Aragorn",1,1);
		Personnage legolas = new Elfe("Legolas",1,1);
		Personnage gimli = new NainGuerrier("Gimli",1,1,5);
		Personnage fili = new NainGuerrierVoleur("Fili",1,1,3,2);
		Personnage frodon = new Hobbit("Frodon",200,100);

		Anneau sauron = frodon.creeAnneauDeSauron();
		sauron.metAuDoigt();


		personnages.add(gandalf);
		personnages.add(aragorn);
		personnages.add(legolas);
		personnages.add(gimli);
		personnages.add(fili);
		personnages.add(frodon);

		for (Personnage heros : personnages) {
			System.out.print(heros.parler() + "! Je suis " + heros.getNom());
			System.out.print(". Ma vitesse est de " + heros.getVitesse() + " km/h. " + heros);
			System.out.println(". Je suis " + (heros.getVisible() ? "visible" : "invisible"));
		}

	}
}
```

2.2. Class Personnage

```java
package heroicfantasy;

public abstract class Personnage {

	private static boolean nbSauron = false;


	public Anneau creeAnneauDeSauron() {
		if (!nbSauron) {
			nbSauron = true;
			return new AnneauDeSauron(this);
		}
		return null;
	}

	private String nom;
	private int pointsVie, x, y, vitesse, sous;
	private boolean visible = true;

	/*
	 * Résumé du rôle de la méthode.
	 * @param n Nom du personnage
	 * @param x position
	 * @param y position
	 *
	 */
	public Personnage(String n, int x, int y, int v) {
		this.nom = n;
		this.x = x;
		this.y = y;
		this.pointsVie = 100;
		this.vitesse = v;
		this.sous = 0;
	}

	public int getSous() {
		return this.sous;
	}

	public void setSous(int s) {
		this.sous = s;
	}

	public int getPointsVie() {
		return this.pointsVie;
	}

	public void setPointsVie(int pv) {
		this.pointsVie = pv;
	}

	public String getNom() {
		return this.nom;
	}

	public int getVitesse() {
		return this.vitesse;
	}

	public void setVisible(boolean b) {
		this.visible = b;
	}

	public boolean getVisible() {
		return this.visible;
	}


	/** Le personnage se deplace dans la direction (dx,dy) durant un temps t. */
	public void seDeplacer(int dx, int dy, int t) {
		this.x = (int) (this.x + dx * this.vitesse * t / Math.sqrt(dx * dx + dy * dy));
		this.y = (int) (this.y + dy * this.vitesse * t / Math.sqrt(dx * dx + dy * dy));
	}

	public abstract String parler();

}
```

2.3. Class Anneau

```java
package heroicfantasy;

public abstract class Anneau {

	protected String nom;
	protected Personnage p;

	public abstract void metAuDoigt();

	public abstract void enleveDuDoigt();
}
```

2.4. Class AnneauDeSauron

```java
package heroicfantasy;

public class AnneauDeSauron extends Anneau {

	AnneauDeSauron(Personnage p) {
		this.nom = "Anneau de Sauron";
		this.p = p;
	}

	public void metAuDoigt() {
		this.p.setVisible(false);
	}

	public void enleveDuDoigt() {
		this.p.setVisible(true);

	}


}
```

3. Exercice 3

On veut pouvoir gérer dans le jeu des objets qui peuvent être des armes, des documents (parchemins, grimoires, messages, etc) ou d'autres objets (bijoux, clés, etc). Un objet a un nom et un prix.

1. On veut pouvoir représenter les personnages humanoides (humains, nains, elfes, hobbits, etc) qui, en plus de pouvoir se déplacer, peuvent acquérir des objets ou s'en séparer. On veut aussi que chaque objet puisse être donné par un humanoide à un autre humanoide. Sans modifier la classe Personnage, proposez du code permettant de représenter les objets et les humanoides.

2. On veut que chaque arme ait une certaine puissance et puisse être utilisée par son propriétaire contre un autre personnage (humanoide ou non). Une arme ne peut pas être utilisée par un personnage autre que son propriétaire. Quand un personnage utilise une arme contre un autre, il retire à ce dernier un nombre de points de vie égal à la puissance de l'arme. Proposez du code pour représenter les armes. Afficher le résultat suivant :

Aragorn acquire Anduril
Frodon à 100 pv
Aragorn attaque "par erreur" Frodon
Frodon à 90 pv

```
System.out.println("Aragorn acquire Anduril");
//code//
System.out.println("Frodon à " + frodon.getPointsVie() + " pv");
System.out.println("Aragorn attaque \"par erreur\" Frodon");
//code//
System.out.println("Frodon à " + frodon.getPointsVie() + " pv");
```

3. On veut que chaque document contienne une certaine quantité de connaissances et puisse être lu par son propriétaire. Un document ne peut pas être lu par un autre personnage que son propriétaire.

Quand un personnage lit un document, son expérience est augmentée d'une quantité égale à la quantité d' expériences du document. Mais cet effet ne peut avoir lieu qu'une seule fois pour un même document : si un personnage lit un document qu'il a déjà lu, son expérience n'est pas modifiée. Proposez du code permettant de représenter les documents. Enrichissez la class Personnage du code suivant :

```
public int getExperience() {
        return this.experience;
}

public void setExperience(int xp) {
        this.experience = xp;
}
```

3.1. Class Testeur

```java
package heroicfantasy;

import java.util.ArrayList;
import java.util.List;

import heroicfantasy.Personnage.Anneau;


public class Testeur {

	public static void main(String[] args) {

		List<Personnage> personnages = new ArrayList<Personnage>();

		Humanoide gandalf = new Humain("Gandalf",1,1);
		Humanoide aragorn = new Humain("Aragorn",1,1);
		Humanoide legolas = new Elfe("Legolas",1,1);
		Humanoide gimli = new NainGuerrier("Gimli",1,1,5);
		Humanoide fili = new NainGuerrierVoleur("Fili",1,1,3,2);
		Humanoide frodon = new Hobbit("Frodon",200,100);


		Anneau sauron = frodon.creeAnneauDeSauron();
		sauron.metAuDoigt();



		personnages.add(gandalf);
		personnages.add(aragorn);
		personnages.add(legolas);
		personnages.add(gimli);
		personnages.add(fili);
		personnages.add(frodon);

		for (Personnage heros : personnages) {
			System.out.print(heros.parler() + "! Je suis " + heros.getNom());
			System.out.print(". Ma vitesse est de " + heros.getVitesse() + " km/h. " + heros);
			System.out.println(". Je suis " + (heros.getVisible() ? "visible" : "invisible"));
		}

		Arme anduril = new Arme("Anduril", 1000, 10);
		System.out.println("Aragorn acquire Anduril");
		aragorn.acquiertObjet(anduril);
		frodon.setPointsVie(100);
		System.out.println("Frodon à " + frodon.getPointsVie() + " pv");
		System.out.println("Aragorn attaque par \"erreur\" Frodon");
		aragorn.attaque(frodon, anduril);
		System.out.println("Frodon à " + frodon.getPointsVie() + " pv");



	}
}
```

3.2. Class Arme

```java
package heroicfantasy;

public class Arme extends Objet {
	private int puissance;

	public Arme(String n, int p, int pui) {
		super(n, p);
		this.puissance = pui;
	}

	public int getPuissance() {
		return this.puissance;
	}
}
```

3.3. Class Objet

```java
package heroicfantasy;

public class Objet {

	private String nom;
	private int prix;

	public Objet(String n, int p) {
		this.nom = n;
		this.prix = p;
	}
}
```

3.4. Class Humanoid

```java
package heroicfantasy;

import java.util.LinkedList;

public abstract class Humanoide extends Personnage {
	private LinkedList<Objet> objets;
	private LinkedList<Document> lus;

	public Humanoide(String n, int x, int y, int v) {
		super(n, x, y, v);
		this.objets = new LinkedList<Objet>();
		this.lus = new LinkedList<Document>();
	}

	public void acquiertObjet(Objet o) {
		this.objets.add(o);
	}

	public void perdObjet(Objet o) {
		this.objets.remove(o);
	}

	public void donnneObjet(Objet o, Humanoide h) {
		if (this.objets.remove(o))
			h.acquiertObjet(o);
	}

	public void attaque(Personnage p, Arme a) {
		if (this.objets.contains(a))
			p.setPointsVie(Math.max(0, p.getPointsVie() - a.getPuissance()));
	}

	public void lit(Document d) {
		if (this.objets.contains(d) && !this.lus.contains(d)) {
			this.setExperience(this.getExperience() + d.getExperience());
			this.lus.add(d);
		}
	}
}
```

3.5. Class Document

```java
package heroicfantasy;

public class Document extends Objet {
	private int experience;

	public Document(String n, int p, int con) {
		super(n, p);
		this.experience = con;
	}

	public int getExperience() {
		return this.experience;
	}
}
```

4. Exercice 4

On veut maintenant représenter les monstres (troll, nazgul, etc.), qui ne peuvent utiliser des objets. L'interface suivante a été définie pour cela, ne peut être modifiée, et doit obligatoirement être utilisée pour représenter les monstres.

```
public interface Monstre{

     public void attaque(Personnage p);
     public int getPuanteur();

}
```

On veut représenter les trolls, qui sont des personnages (pouvant se déplacer) et en même temps des monstres. Un troll a une certaine force et peut attaquer un personnage en lui faisant perdre un nombre de points de vie égal à sa force plus le dixième de sa puanteur (disons qu'il à une puanteur à 200). Mais un troll n'attaque jamais un autre troll. Proposez du code permettant de représenter les trolls. Le troll dit "Hiark" quand il parle.

4.1. Class Troll

```java
package heroicfantasy;

public class Troll extends Personnage implements Monstre {
	private int force;

	public Troll(String n, int x, int y, int v, int force) {
		super(n, x, y, v);
		this.force = force;
	}

	public int getPuanteur() {
		return 200;
	}

	public void attaque(Personnage p) {
		if (!(p instanceof Troll))
			p.setPointsVie(Math.max(0, p.getPointsVie() - this.force - this.getPuanteur() / 10));
	}

	public String parler() {
		return "Hiark";
	}
}
```

TP - Swing

Site: AMIO-FIT
Cours: Session février à avril 2022 - 16 séances
Livre: TP - Swing

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:27

Description

Réaliser un jeu sur le thème du Seigneur des anneaux (suite de TP, rester dans le package heroicfantasy).

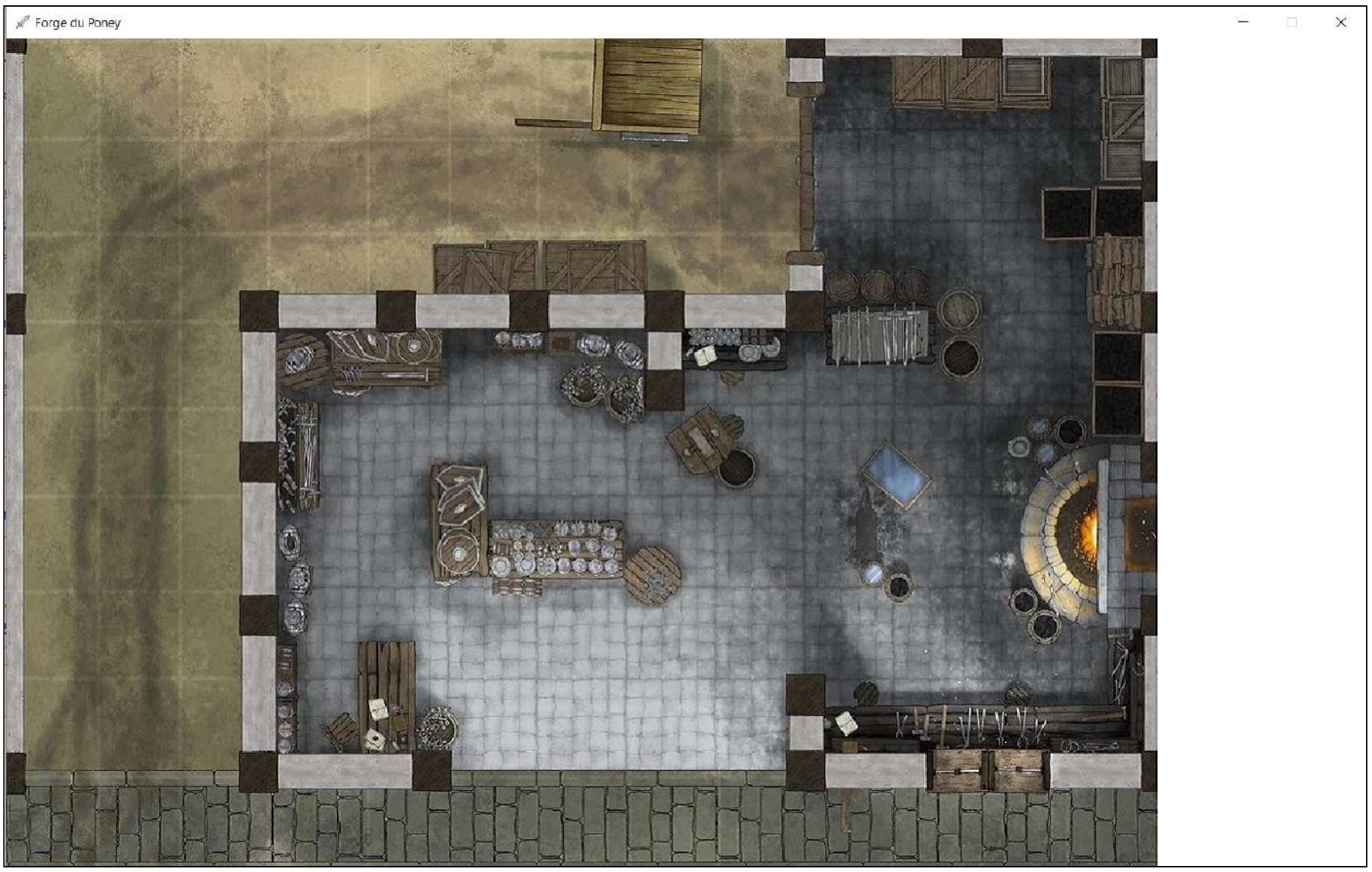
Table des matières

1. Exercice 1

Créer une fenêtre en utilisant le composant SWING avec les caractéristiques ci-dessous. Votre fenêtre devra être géré dans la `classe Scene` et devra être lancer à partir de la `class Run` (Class Testeur pour test et Run pour app final) :

- fenêtre de taille 1300 x 800
- Titre "Forge du Poney" (en argument lors de l'instanciation de l'objet `scene`)
- Icone de fenêtre utiliser le fichier **epee.png**
- Ajouter une image de fond **forge.jpg**
- Votre fenêtre ne doit pas redimensionnable
- Vos ressources fichiers images doivent avoir la structure suivante à la racine de votre projet :

Vous devez obtenir le résultat suivant :



1.1. Solution

```java
package heroicfantasy;

import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.ImageIcon;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class Scene extends JFrame {

	// Dimension fenetre
	private int weight = 1300;
	private int height = 800;

	// constructeur
	public Scene(String title) {

		// Titre fenetre
		super(title);

		// dimension fenetre
		this.setSize(new Dimension(weight, height));

		// icone fenetre
		ImageIcon image = new ImageIcon(".\\static\\img\\icon\\epee.png");
		this.setIconImage(image.getImage());

		// Position fenetre centré, aligné haut
		Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
		this.setLocation(dim.width / 2 - getWidth() / 2, 0);

		// Background
		this.setContentPane(new JLabel(new ImageIcon(".\\static\\img\\scenes\\forge.jpg")));

		// ne pas crop l'image
		this.pack();

		// ne pas redimensioner
		this.setResizable(false);

		// Afficher
		this.setVisible(true);
	}

}
```

Run :

//fenetre

```
Scene forge = new Scene("Forge du Poney");
```

2. Exercice 2

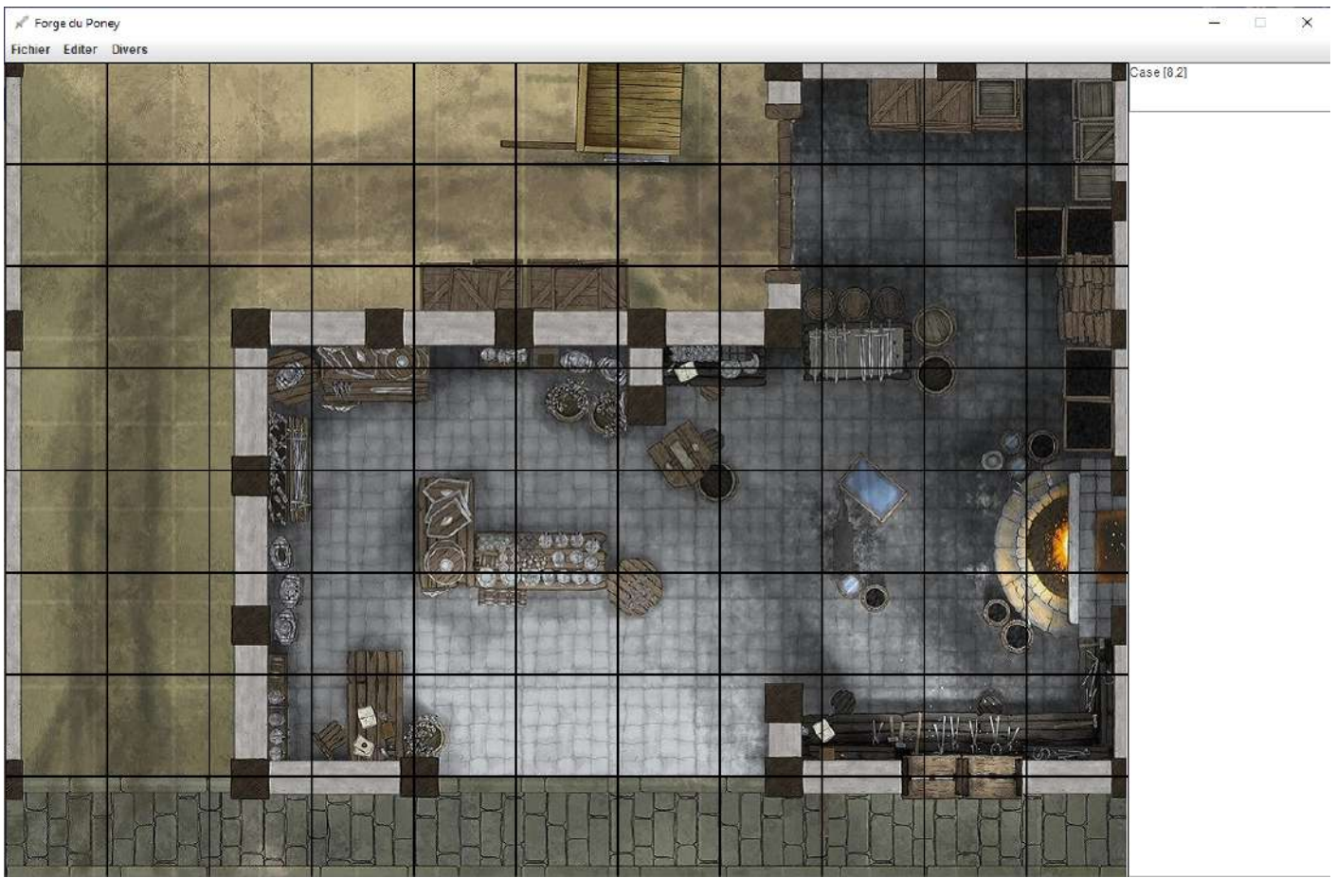
Nous souhaitons réaliser un menu et un quadrillage sur l'image et réserver la partie droite de l'IHM (zone blanche) pour les actions futur du jeu (bouton, affichage d'information, etc.).

Chaque case du quadrillage, appelé cellule, est considéré comme une zone cliquable.

Réaliser le développement suivant :

- Compléter la `class Scene` avec la `methode Grid`
  - Une cellule est un carré de coté 100
  - Le cadrillage doit comporter 11 colonnes sur la largeur et 8 colonnes sur la hauteur
    - La première cellule en haut a gauche à la coordonnées 0,0 et la dernière cellule en bas à droite la coordonnées 10,7
  - Chaque cellule est cliquable et doit renvoyer l'information suivante : Case [0,1] par exemple si l'on clique sur la cellule de coordonnées 0,1.
  - Réaliser une zone d'information de type `JTextArea` avec les dimensions 200 de large et 50 de haut sur la partie droite de l'application
    - Faire afficher le résultat de la cellule cliqué dans cette zone d'information
- Réaliser la `class Menu`
  - Intégrer le menu vu dans le cours à votre application `class Menu`

Résultat attendu :



2.1. Class Menu

```java
package heroicfantasy;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JRadioButtonMenuItem;

public class Menu extends JMenuBar {

	public Menu() {

		// Listener générique qui affiche l'action du menu utilisé
		ActionListener afficherMenuListener = new ActionListener() {
			public void actionPerformed(ActionEvent event) {
				System.out.println("Elément de menu [" + event.getActionCommand() + "] utilisé.");
			}
		};

		// Création du menu Fichier
		JMenu fichierMenu = new JMenu("Fichier");
		JMenuItem item = new JMenuItem("Nouveau", 'N');
		item.addActionListener(afficherMenuListener);
		fichierMenu.add(item);
		item = new JMenuItem("Ouvrir", 'O');
		item.addActionListener(afficherMenuListener);
		fichierMenu.add(item);
		item.addActionListener(afficherMenuListener);
		fichierMenu.add(item);
		item = new JMenuItem("Sauver", 'S');
		item.addActionListener(afficherMenuListener);
		fichierMenu.insertSeparator(1);
		fichierMenu.add(item);
		item = new JMenuItem("Quitter");
		item.addActionListener(afficherMenuListener);
		fichierMenu.add(item);

		// Création du menu Editer
		JMenu editerMenu = new JMenu("Editer");
		item = new JMenuItem("Copier");
		item.addActionListener(afficherMenuListener);

		editerMenu.add(item);
		item = new JMenuItem("Couper");
		item.addActionListener(afficherMenuListener);

		editerMenu.add(item);
		item = new JMenuItem("Coller");
		item.addActionListener(afficherMenuListener);

		editerMenu.add(item);

		// Création du menu Divers
		JMenu diversMenu = new JMenu("Divers");
		JMenu sousMenuDiver1 = new JMenu("Sous menu 1");

		item.addActionListener(afficherMenuListener);
		item = new JMenuItem("Sous menu 1 1");
		sousMenuDiver1.add(item);
		item.addActionListener(afficherMenuListener);
		JMenu sousMenuDivers2 = new JMenu("Sous menu 1 2");
		item = new JMenuItem("Sous menu 1 2 1");
		sousMenuDivers2.add(item);
		sousMenuDiver1.add(sousMenuDivers2);

		diversMenu.add(sousMenuDiver1);
		item = new JCheckBoxMenuItem("Validé");
		diversMenu.add(item);
		item.addActionListener(afficherMenuListener);
		diversMenu.addSeparator();
		ButtonGroup buttonGroup = new ButtonGroup();
		item = new JRadioButtonMenuItem("Cas 1");
		diversMenu.add(item);
		item.addActionListener(afficherMenuListener);
		buttonGroup.add(item);
		item = new JRadioButtonMenuItem("Cas 2");
		diversMenu.add(item);
		item.addActionListener(afficherMenuListener);
		buttonGroup.add(item);
		diversMenu.addSeparator();
		diversMenu.add(item = new JMenuItem("Autre", new ImageIcon(".\\static\\img\\icon\\epee.png")));
		item.addActionListener(afficherMenuListener);

		// ajout des menus à la barre de menus
		this.add(fichierMenu);
		this.add(editerMenu);
		this.add(diversMenu);
	}
}
```

2.2. Class Scene

```java
package heroicfantasy;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.LineBorder;

public class Scene extends JFrame {

	// Dimension fenetre
	private int weight = 1300;
	private int height = 800;

	// constructeur
	public Scene(String title) {

		// Titre fenetre
		super(title);

		// dimension fenetre
		this.setSize(new Dimension(weight, height));

		// icone fenetre
		ImageIcon image = new ImageIcon(".\\static\\img\\icon\\epee.png");
		this.setIconImage(image.getImage());

		// Position fenetre centré, aligné haut
		Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
		this.setLocation(dim.width / 2 - getWidth() / 2, 0);

		// Background
		this.setContentPane(new JLabel(new ImageIcon(".\\static\\img\\scenes\\forge.jpg")));

		// Ajout Menu
		this.setJMenuBar(new Menu());

		// Ajout Grid
		this.Grid();

		// ne pas crop l'image
		this.pack();

		// ne pas redimensioner
		this.setResizable(false);
		// Afficher
		this.setVisible(true);
	}

	private void Grid() {
		// Zone de texte
		int text_w = 200;
		int text_h = 50;

		// taille cellule
		int cellule = 100;

		// Zone d'information
		JTextArea textArea = new JTextArea();
		JScrollPane scrollPane = new JScrollPane(textArea);
		scrollPane.setBounds(weight - text_w, 0, text_w, text_h);
		this.add(scrollPane);
		// Ajout des cases bouton
		// ligne
		for (int i = 0; i < weight - text_w; i += cellule) {
			// colonne
			for (int h = 0; h < height; h += cellule) {
				// créer un bouton
				JButton btn = new JButton();
				// btn.setC
				// definir la taille
				btn.setSize(cellule, cellule);
				// definir border
				btn.setBorder(new LineBorder(Color.BLACK));
				// définir la position du bouton
				btn.setBounds(i, h, cellule, cellule);
				// ajouter event
				btn.addActionListener(new ActionListener() {
					public void actionPerformed(ActionEvent event) {
						int caseX = btn.getBounds().x / cellule;
						int caseY = btn.getBounds().y / cellule;
						String message = "Case [" + caseX + "," + caseY + "]";
						textArea.setText(message);
					}
				});
				// rendre le bouton transparent
				btn.setOpaque(false);
```

```
                                // enlever la zone de contenu
                                btn.setContentAreaFilled(false);
                                btn.setBorderPainted(true);
                                // ajout bouton et text au frame
                                this.add(btn);

                        }
                }
        }
}
```

3. Exercice 3

En reprenant les derniers TP sur l'abstraction et l'héritage. Créer les trois personnages suivant :

- Aragorn en position 1,0
- Legolas en position 5,5
- Nazgul en position 7,4

Afficher les personnages sur la scène et faite en sorte de les faire parler quand vous cliquer sur eux. Le message doit s'afficher dans la zone texte créer dans l'exercice précédent. Chaque personnage a un skin. Les ressources graphiques sont **disponible ici**.

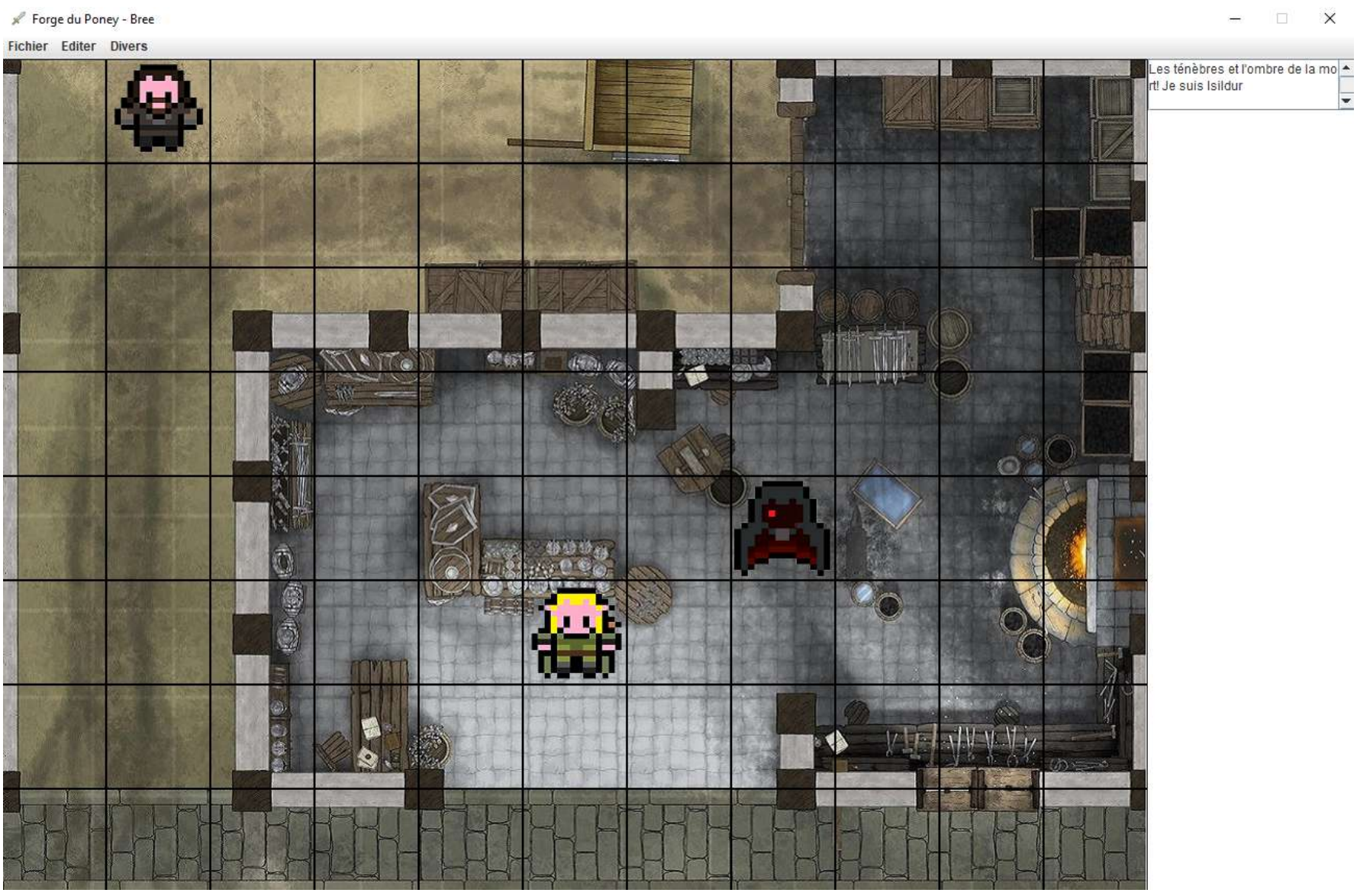
Le Nazgul est Isildur, c'est un monstre qui se comporte comme un guerrier et dit "Les ténèbres et l'ombre de la mort". Quand il attaque il absorbe un dixième de la vie de son adversaire. Sa vitesse est de 5 et passe a 27 si il chevauche un "Fellbeast" qui se comporte comme un familier qui dépend de la class Individu (refactoriser Personnage pour que la class Individu devienne Parente).

Tout les personnages peuvent avoir une monture, la vitesse du personnage est égale à la vitesse de la monture - 10%. Le familier qui à un cavalier perd aussi 10% de vitesse.

Dans ce scenario, Isildur est représenté après avoir descendu de sa monture.

- Trouver une solution qui permet d'accéder à tous les personnages instanciés du jeu.
- Trouver une solution qui permet d'accéder à tous les personnages d'une scène.
- Réaliser les classes et méthodes correspondantes.

Résultat attendu :



3.1. Class Individu

```java
package heroicfantasy;

import java.util.ArrayList;
import java.util.List;

public abstract class Individu {

	static List<Individu> individus = new ArrayList<Individu>();

	private String nom, skin;
	private int pointsVie, experience, x, y;

	protected int vitesse;

	public Individu(String n, int x, int y, int v) {
		this.nom = n;
		this.x = x;
		this.y = y;
		this.setVitesse(v);
		this.setExperience(0);
		this.setPointsVie(100);
		individus.add(this);

	}

	public int getX() {
		return x;
	}

	public void setX(int x) {
		this.x = x;
	}

	public int getY() {
		return y;
	}

	public void setY(int y) {
		this.y = y;
	}

	public int getPointsVie() {
		return this.pointsVie;
	}

	public void setPointsVie(int pv) {
		this.pointsVie = pv;
	}

	public String getNom() {
		return this.nom;
	}

	public int getVitesse() {
		return this.vitesse;
	}

	public void setVitesse(int v) {
		this.vitesse = v;
	}
	public int getExperience() {
		return this.experience;
	}

	public void setExperience(int xp) {
		this.experience = xp;
	}

	public void setSkin(String skin) {
		this.skin = skin;
	}

	public String getSkin() {
		return this.skin;
	}



	/** Le personnage se deplace dans la direction (dx,dy) durant un temps t. */
	public void seDeplacer(int dx, int dy, int t) {
		this.x = (int) (this.x + dx * this.vitesse * t / Math.sqrt(dx * dx + dy * dy));
		this.y = (int) (this.y + dy * this.vitesse * t / Math.sqrt(dx * dx + dy * dy));
	}

	public abstract String parler();

}
```

3.2. Class Run

```java
package heroicfantasy;

import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;

public class Run {

	public static void main(String[] args) {


		Humain aragorn = new Humain("Aragorn",1,0);
		Elfe legolas = new Elfe("Legolas",5,5);
		Nazgul isildur = new Nazgul("Isildur", 7, 4, 50);
		Familier fellbeast = new Fellbeast("Fellbeast",7,4,30,100);

		//Monture
		isildur.setMonture(fellbeast);
		System.out.println(isildur.getVitesse());
		System.out.println(fellbeast.getVitesse());
		//Descendre monture
		isildur.setMonture(null);


		aragorn.setSkin(Scene.assetPath + "heros\\aragorn.png");
		legolas.setSkin(Scene.assetPath + "heros\\legolas.png");
		isildur.setSkin(Scene.assetPath + "heros\\nazgul.png");

		Humanoide frodon = new Hobbit("Frodon",8,1);

		Anneau sauron = frodon.creeAnneauDeSauron();
		sauron.metAuDoigt();
		frodon.getVisible();

		//Start scene
		Scene forge = new Scene("Bree - Forge du Poney", "forge");
		forge.addIndividu(aragorn);
		forge.addIndividu(legolas);
		forge.addIndividu(isildur);
		// Ajout Grid
		forge.setGrid();

		// Afficher Scene
		forge.setVisible(true);
	}

}
```

3.3. Class Scene

```java
package heroicfantasy;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.LineBorder;

public class Scene extends JFrame {

	// Dimension fenetre
	private int weight = 1300;
	private int height = 800;
	// taille cellule
	private int cellule = 100;
	public static String assetPath = ".\\static\\img\\";

	// stock les personnages présent dans la scene
	static List<Individu> individus = new ArrayList<Individu>();

	// constructeur
	public Scene(String title, String bg) {

		// Titre fenetre
		super(title);

		// dimension fenetre
		this.setSize(new Dimension(weight, height));

		// icone fenetre
		ImageIcon image = new ImageIcon(assetPath + "icon\\epee.png");
		this.setIconImage(image.getImage());

		// Position fenetre centré, aligné haut
		Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
		this.setLocation(dim.width / 2 - getWidth() / 2, 0);

		// Background
		this.setContentPane(new JLabel(new ImageIcon(assetPath + "scenes\\" + bg + ".jpg")));

		// Ajout Menu
		this.setJMenuBar(new Menu());

		// ne pas crop l'image
		this.pack();

		// ne pas redimensioner
		this.setResizable(false);

	}

	public void setGrid() {
		// Zone de texte
		int text_w = 200;
		int text_h = 50;

		// Zone d'information
		JTextArea textArea = new JTextArea();
		textArea.setLineWrap(true);
		JScrollPane scrollPane = new JScrollPane(textArea, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
				JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

		scrollPane.setBounds(weight - text_w, 0, text_w, text_h);

		this.add(scrollPane);
		// Ajout des cases bouton
		// ligne
		for (int i = 0; i < weight - text_w; i += cellule) {
			// colonne
			for (int h = 0; h < height; h += cellule) {
				// créer un bouton
				JButton btn = new JButton();
				// btn.setC
				// definir la taille
				btn.setSize(cellule, cellule);
				// definir border
				btn.setBorder(new LineBorder(Color.BLACK));
				// définir la position du bouton
				btn.setBounds(i, h, cellule, cellule);
				// ajouter event
				btn.addActionListener(new ActionListener() {
					public void actionPerformed(ActionEvent event) {
						int caseX = btn.getBounds().x / cellule;
						int caseY = btn.getBounds().y / cellule;
						String message = "Case [" + caseX + "," + caseY + "]";
```

```
                                        for (Individu individu : individus) {

                                                if (caseX == individu.getX() && caseY == individu.getY()) {

                                                        message = individu.parler() + "! Je suis " + individu.getNom();
                                                }
                                        }

                                        textArea.setText(message);
                                }
                        });
                        // rendre le bouton transparent
                        btn.setOpaque(false);
                        // enlever la zone de contenu
                        btn.setContentAreaFilled(false);
                        btn.setBorderPainted(true);
                        // ajout bouton et text au frame
                        this.add(btn);

                }
        }
    }

    public void addIndividu(Individu i) {

            individus.add(i);
            JLabel individu = new JLabel(new ImageIcon(i.getSkin()));
            individu.setBounds(i.getX() * this.cellule, i.getY() * this.cellule, this.cellule, this.cellule);
            this.add(individu);

    }
}
```

3.4. Class Personnage

```java
package heroicfantasy;

import java.util.ArrayList;
import java.util.List;

public abstract class Personnage extends Individu {

	private static boolean nbSauron = false;


	private int sous;

	private boolean visible = true;
	private Familier monture = null;



	public Personnage(String n, int x, int y, int v) {
		super(n, x, y, v);
		this.setSous(0);

	}

	public int getSous() {
		return this.sous;
	}

	public void setSous(int s) {
		this.sous = s;
	}


	public void setVisible(boolean b) {
		this.visible = b;
	}

	public boolean getVisible() {
		return this.visible;
	}

	public int getVitesse() {
		if(this.monture != null) {
			return (int) Math.round(this.monture.getVitesse());
		}else {
			return this.vitesse;
		}
	}



	public Anneau creeAnneauDeSauron() {
		if (!nbSauron) {
			nbSauron = true;
			return new AnneauDeSauron(this);
		}
		return null;
	}

	public Familier getMonture() {
		return this.monture;
	}

	public void setMonture(Familier f) {
		if (f == null) {
			this.monture.setCavalier(null);
			this.monture = null;
		}else {
			this.monture = f;
			f.setCavalier(this);

		}

	}


}
```

3.5. Class Familier

```java
package heroicfantasy;

public abstract class Familier extends Individu {


	public Personnage cavalier = null;

	public Familier(String n, int x, int y, int v) {
		super(n, x, y, v);
	}

	protected void setCavalier(Personnage p) {

		this.cavalier = p;

	}

	public int getVitesse() {
		if(this.cavalier != null) {
			return (int) Math.round(this.vitesse * 0.9);
		}else {
			return this.vitesse;
		}
	}

}
```

3.6. Class Nazgul

```java
package heroicfantasy;

public class Nazgul extends Personnage implements Guerrier, Monstre {
	private int force;

	public Nazgul(String n, int x, int y, int force) {
		super(n, x, y, 5);
		this.force = force;
	}

	public int getForce() {
		return this.force;
	}

	public void attaque(Individu i) {
			i.setPointsVie(Math.max(0, i.getPointsVie() - this.force));
			this.setPointsVie(Math.max(0, this.getPointsVie() + (i.getPointsVie() / 10)));
	}

	public String parler() {
		return "Les ténèbres et l'ombre de la mort";
	}


	public int getPuanteur() {
		// TODO Auto-generated method stub
		return 0;
	}
}
```

3.7. Class Fellbeast

```java
package heroicfantasy;

public class Fellbeast extends Familier implements Monstre {
	private int force;

	public Fellbeast(String n, int x, int y, int v, int force) {
		super(n, x, y, 30);
		this.force = force;
	}

	public int getPuanteur() {
		return 50;
	}

	public void attaque(Individu i) {
			i.setPointsVie(Math.max(0, i.getPointsVie() - this.force));
	}

	public String parler() {
		return "Coaaxxxxxx";
	}
}
```

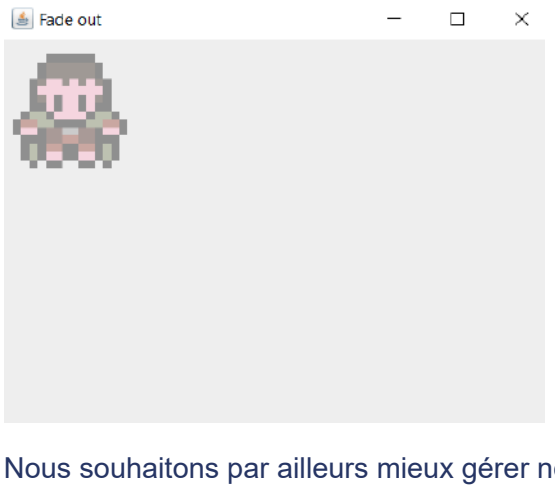
4. Exercice 4

Nous souhaiterions que le Nazgul, rend visible le porteur de l'anneau si il est sur la même scène.

- Ajouter frodon sur la scene en [8, 1], il porte l'anneau il est donc invisible
  - On le representera avec une opacité de 50%
- Par contre si le Nazgul est dans la même scene il apparaitra avec effet opacité de 0 à 100.

Attention vous devez laisser vos méthodes concernant la visibilité dans Personnage.

Pour gérer l'opacité sur une image vous pouvez vous inspirer de **l'exemple ici**.



Nous souhaitons par ailleurs mieux gérer notre code par package, déplacer vos classes graphiques dans le nouveau package heroicfantasy_interface :

- Scene, Menu, FadeIn

4.1. Class Scene

```java
package heroicfantasy_interface;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.LineBorder;

import heroicfantasy.AnneauDeSauron;
import heroicfantasy.Individu;
import heroicfantasy.Nazgul;

public class Scene extends JFrame {

	// Dimension fenetre
	private int weight = 1300;
	private int height = 800;
	// taille cellule
	private int cellule = 100;
	public static String assetPath = ".\\static\\img\\";
	// Zone de texte
	private int text_w = 200;
	private int text_h = 200;

	// stock les personnages présent dans la scene
	static List<Individu> individus = new ArrayList<Individu>();

	// constructeur
	public Scene(String title, String bg) {

		// Titre fenetre
		super(title);

		// dimension fenetre
		this.setSize(new Dimension(weight, height));

		// icone fenetre
		ImageIcon image = new ImageIcon(assetPath + "icon\\epee.png");
		this.setIconImage(image.getImage());

		// Position fenetre centré, aligné haut
		Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
		this.setLocation(dim.width / 2 - getWidth() / 2, 0);

		// Background
		this.setContentPane(new JLabel(new ImageIcon(assetPath + "scenes\\" + bg + ".jpg")));

		// Ajout Menu
		this.setJMenuBar(new Menu());

		// ne pas crop l'image
		this.pack();

		// ne pas redimensioner
		this.setResizable(false);

	}

	public void setGrid() {


		// Zone d'information
		JTextArea textArea = new JTextArea();
		textArea.setLineWrap(true);
		JScrollPane scrollPane = new JScrollPane(textArea, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
				JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

		//padding
		scrollPane.setBorder(BorderFactory.createCompoundBorder(
				textArea.getBorder(),
			BorderFactory.createEmptyBorder(5, 5, 5, 5)));

		scrollPane.setBounds(weight - text_w, 0, text_w, text_h);

		this.add(scrollPane);
		// Afficher individu
		afficheIndividu();
		// Ajout des cases bouton
		// ligne
		for (int i = 0; i < weight - text_w; i += cellule) {
			// colonne
			for (int h = 0; h < height; h += cellule) {
				// créer un bouton
				JButton btn = new JButton();
```

```
                // btn.setC
                // definir la taille
                btn.setSize(cellule, cellule);
                // definir border
                btn.setBorder(new LineBorder(Color.BLACK));
                // définir la position du bouton
                btn.setBounds(i, h, cellule, cellule);

                // ajouter event
                btn.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent event) {
                                int caseX = btn.getBounds().x / cellule;
                                int caseY = btn.getBounds().y / cellule;
                                String message = "Case [" + caseX + "," + caseY + "]";

                                for (Individu individu : individus) {

                                        if (caseX == individu.getX() && caseY == individu.getY()) {

                                                message = individu.parler() + "! Je suis " + individu.getNom();
                                        }
                                }

                                textArea.setText(message);
                        }
                });
                // rendre le bouton transparent
                btn.setOpaque(false);
                // enlever la zone de contenu
                btn.setContentAreaFilled(false);
                btn.setBorderPainted(true);
                // ajout bouton et text au frame
                this.add(btn);

                        }
                }
        }

        public void addIndividu(Individu i) {

                i.setScene(this);
                individus.add(i);

        }

        private void afficheIndividu() {

                for (Individu i : individus) {

                        JLabel individu = new JLabel(new ImageIcon(i.getSkin()));

                        if (AnneauDeSauron.porteurAnneauDeSauron == i && AnneauDeSauron.porteurAnneauDeSauron.getVisible() == false) {
                                if (individus.stream().filter(o -> o instanceof Nazgul).findFirst().isPresent()) {
                                        individu = new FadeIn((i.getSkin()));
                                }else {
                                        individu = new FadeIn((i.getSkin()), 0.5f);
                                }

                        }

                        individu.setBounds(i.getX() * this.cellule, i.getY() * this.cellule, this.cellule, this.cellule);

                        this.add(individu);
                }

        }
}
```

4.2. Class FadeIn

```java
package heroicfantasy_interface;

import java.awt.AlphaComposite;
import java.awt.Graphics;

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.Timer;

public class FadeIn extends JLabel implements ActionListener {

	Image imagem;
	Timer timer;
	private float alpha = 0f;

	public FadeIn(String path) {
		imagem = new ImageIcon(path).getImage();
		timer = new Timer(100, this);
		timer.start();
	}

	// here you define alpha 0f to 1f
	public FadeIn(String path, float alpha) {
		imagem = new ImageIcon(path).getImage();
		this.alpha = alpha;

	}

	public void paintComponent(Graphics g) {
		super.paintComponent(g);

		Graphics2D g2d = (Graphics2D) g;

		/*
		 * fond gradient g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
		 * RenderingHints.VALUE_RENDER_QUALITY); int w = getWidth(), h = getHeight();
		 * Color color1 = Color.RED; Color color2 = Color.YELLOW; GradientPaint gp = new
		 * GradientPaint(0, 0, color1, w, h, color2); g2d.setPaint(gp); g2d.fillRect(0,
		 * 0, w, h);
		 */
		g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha));

		g2d.drawImage(imagem, 0, 0, null);

	}

	public void actionPerformed(ActionEvent e) {
		alpha += 0.05f;
		if (alpha > 1) {
			alpha = 1;
			timer.stop();
		}
		repaint();
	}
}
```

4.3. Class AnneauDeSauron

```java
package heroicfantasy;

public class AnneauDeSauron extends Anneau {

	public static Personnage porteurAnneauDeSauron = null;

	AnneauDeSauron(Personnage p) {
		this.nom = "Anneau de Sauron";
		porteurAnneauDeSauron = p;
	}

	public void metAuDoigt() {
		porteurAnneauDeSauron.setVisible(false);
	}

	public void enleveDuDoigt() {
		porteurAnneauDeSauron.setVisible(true);

	}


}
```