Document de travail

ENSEA

Beyond Engineering

PROGRAMMATION ORIENTÉE OBJET EN JAVA

DTI_

Sujets de Travaux Dirigés machines - 14 h

2ième année

Antoine Tauvel

Hiver 2024-2025

Contents

Les TDs et TPs sont réalisés avec vos PC personnels en utilisant IntelliJ, un IDE commerciale de développement disponible gratuitement avec un compte éducation. Un IDE (*Integrated Debug Environnment*) est une suite de développement, un outil dont la maîtrise est indispensable à un informaticien.
IntelliJ est lui même codé en Java, il peut donc s'éxécuter sur Windows, Linux et Max OS. Vous **devez** l'avoir installé **avant** la première séance de TD.
Attention ! L'ensemble de ce dispositif pédagogique repose sur la classe inversé. Si vous ne faites pas le travail préparatoire, vous ne retirerez pas les bonnes compétences de ces séances.
L'enseignant responsable de la séance de TP ou de TD peut refuser l'accès à la séance à un étudiant n'ayant pas fait le travail préparatoire.

The tutorials and practical sessions are conducted using your personal computers with IntelliJ, a commercial development IDE available for free with an educational account. An IDE (*Integrated Development Environment*) is a development suite, a tool whose mastery is essential for a computer scientist.
IntelliJ itself is coded in Java, so it can run on Windows, Linux, and Mac OS. You **must** have it installed **before** the first tutorial session.
Attention! This entire educational setup is based on the flipped classroom model. If you do not complete the preparatory work, you will not gain the necessary skills from these sessions.
The instructor responsible for the practical or tutorial session may deny access to a student who has not completed the preparatory work.

1 TD Guidé numéro 1 : introduction au langage

1.1 Exercice 1 : différences entre Java et le C.

Soit le code Java ci-dessous. Ce code affiche l'ensemble des paramètres passé au programme, un par ligne. Nous allons utiliser ce code très simple pour mettre en évidence quelques différences entre Java et le C. Pour passer des paramètres, vous pouvez bien entendu utiliser la console (`java Main riri fifi loulou`, mais le plus simple est de donner les paramètres à IntelliJ. Pour cela, aller dans le menu Run / Edit Configuration comme dans la capture ci-dessus.

Given the Java code below. This code displays all the parameters passed to the program, one per line. We will use this very simple code to highlight some differences between Java and C. To pass parameters, you can of course use the console (`java Main riri fifi loulou`), but the easiest way is to give the parameters to IntelliJ. To do this, go to the Run / Edit Configuration menu as shown in the screenshot above.

```
1 public class Main{
2     public static void main(String[] args){
3         for (int i=0;i<args.length;i++){
4             System.out.println("args["+i+"] = "+args[i]);
5         }
6     }
7 }
```

Listing 1: Notre premier programme : guère mieux qu'un Hello world...

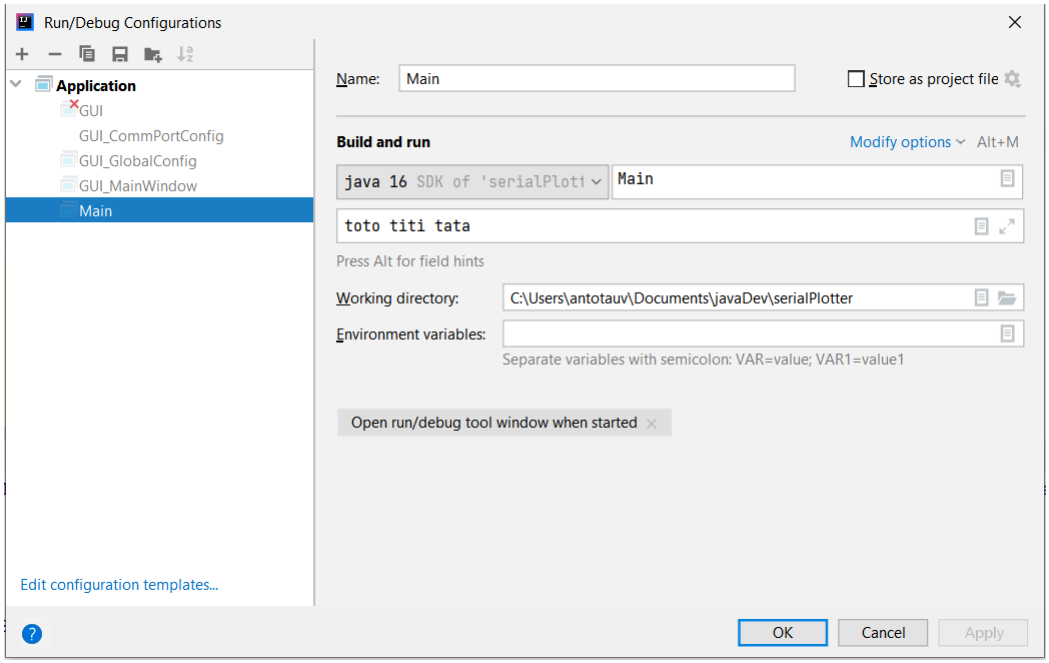

Figure 1.1: Comment fixer les arguments du programme dans IntelliJ

Saisissez, compilez et exécutez le programme. Répondez ensuite aux questions suivantes :

- La variable args est de type String[]. Il existe donc un type (en fait une classe) pour les chaîne de caractère en Java. Est-ce le cas en C ?
- Comment définir un tableau en C ? Et en Java ? Laquelle de ces syntaxes est la plus naturelle ?
- Comment passent-on les paramètres en C. Quelles différences notables voyez-vous ici ?
- Quelle syntaxe permet de déterminer la taille du tableau. Est-ce possible en C ? *What syntax is used to calculate array's size ? Is it available in C ?*
- A quoi correspond l'opération "+" appliqué à une chaine de caractère (ligne 5). En quoi est-ce intéressant ?
- A quoi correspond les syntaxes "public" et "static" définissant la méthode main ? A quoi correspond le mot clé static en C ? Et en Java (nous réapprafondirons cette question dans la séquence 2).
- Recodez la boucle for comme indiqué ci-dessous. L'usage d'un tel itérateur (c'est le terme technique) est il possible en C ?
- Enfin, utilisez le dernier code ci dessous.

Enter, compile, and run the program. Then answer the following questions:

- The variable `args` is of type `String[]`. Therefore, there is a type (actually a class) for strings in Java. Is this the case in C?
- How do you define an array in C? And in Java? Which of these syntaxes is more natural?
- How are parameters passed in C? What notable differences do you see here?
- What syntax allows you to determine the size of the array? Is this possible in C?
- What does the operation `"+"` applied to a string (line 5) correspond to? Why is this interesting?
- What do the syntaxes `public` and `static` defining the `main` method correspond to? What does the keyword `static` correspond to in C? And in Java (we will delve deeper into this question in sequence 2).
- Rewrite the `for` loop as indicated below. Is the use of such an iterator possible in C?
- Finally, use the last code below.

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

```
        for (String string : args){
            System.out.println(string);
        }
    }
}
```

```
public class Main{
    public static void main(String[] args){
                    System.out.println(args);
    }
}
```

1.2 Notre première classe. Encapsulation.

Fabriquez le corps d'une classe Student ayant les attributs suivants :

- Une chaîne de caractère privée finale contenant le nom de l'élève (name).
- Un entier privée représentant son âge (age).

Codez deux constructeurs possible. L'un passe les deux paramètres, l'autre ne passe que le nom. Dans ce cas l'age à une valeur par défaut de 20.
Codez une méthode String toString() qui renvoie une chaîne de caractère contenant le nom et l'age de l'élève.
Testez votre travail à l'aide du code ci-dessous.

Create the body of a `Student` class with the following attributes:

- A private final string containing the student's name (`name`).
- A private integer representing their age (`age`).

Implement two possible constructors. One takes both parameters, the other only takes the name. In this case, the age defaults to 20.
Implement a method `String toString()` that returns a string containing the student's name and age.
Test your work with the code below.

```
public class Main {

    public static void main(String[] args) {
        // write your code here
        Student[] group = new Student[5];
        group[0]=new Student("Riri",12);
        group[1]=new Student("Fifi",10);
        group[2]=new Student("Loulou",81);
```

```
        for (Student student : group){
            System.out.println(student);
        }
    }
}
```

Listing 2: Un moment émouvant, la création de nos premières classes.

- Combien y-a-t-il d'instance de la classe Student ?
- Combien y-a-t-il de référence de la classe Student ?
- Sur quoi pointent group[3] et group[4] ?
- Faites un schéma représentant l'état de la mémoire.

- How many instances of the `Student` class are there?
- How many references to the `Student` class are there?
- What do `group[3]` and `group[4]` point to?
- Draw a diagram representing the state of memory.

Dans la classe Student, codez une méthode `void vieillir()` qui ajoute une année à l'âge de l'élève.
Testez cette méthode en ajoutant une boucle for dans le programme principale faisant viellir tous les membres de StudentGroup.
Quel est l'intérêt de l'encapsulation de données ?

In the `Student` class, implement a method `void vieillir()` that adds one year to the student's age.
Test this method by adding a `for` loop in the main program that ages all members of `StudentGroup`.
What is the benefit of data encapsulation?

```
public class Main {

    public static void main(String[] args) {
        // write your code here
        Student[] group = new Student[5];
        group[0]=new Student("Riri",12);
        group[1]=new Student("Fifi",10);
```

```
        group[2]=new Student("Loulou",81);
        group[1]=group[2];
        group[3]=group[1];
        for (Student student : group){
            System.out.println(student);
        }
    }
}
```

Listing 3: Le garbage collector rentre en scène.

Avant d'exécuter ce programme dans notre IDE, faites un schéma de l'usage de la mémoire à la fin de ce programme.
Testez le code.
Que va-t-il arriver aux classes orphelines (classe sans référence) ?

Before running this program in our IDE, draw a diagram of the memory usage at the end of this program.
Test the code.
What will happen to the orphan classes (classes without references)?

1.3 A faire s'il reste du temps.

- Créez une classe StudentLabGrade qui contient un binôme (tableau) d'élève, ainsi qu'une note. Le binôme d'élève est final.
- Créez un constructeur pour cette classe
- Au sein de la classe Student, codez une méthode `double calculateMean(StudentLabGrade[] studentLabGrades);` qui renvoie la moyenne calculé du tableau de note *en ne tenant compte que des notes de l'élève !* .
- Testez votre méthode avec le code ci-dessous (la moyenne de Riri doit être de 14 et celle de Fifi de 15).

- Create a class `StudentLabGrade` that contains a pair (array) of students, as well as a grade. The pair of students is final.
- Create a constructor for this class.
- Within the `Student` class, implement a method `double calculateMean(StudentLabGrade[] studentLabGrades);` that returns the calculated average of the grade array *only taking into account the grades of the student!*.
- Test your method with the code below (Riri's average should be 14 and Fifi's should be 15).

```java
public class Main {

    public static void main(String[] args) {
        // write your code here
        Student[] group = new Student[5];
        group[0]=new Student("Riri",12);
        group[1]=new Student("Fifi",10);
        group[2]=new Student("Loulou",81);

        StudentLabGrade[] studentLabGrades= new StudentLabGrade[2];
        studentLabGrades[0] = new StudentLabGrade(group[0],group[1],14.0);
        studentLabGrades[1] = new StudentLabGrade(group[1],group[2],16.0);

        System.out.println("Moyenne de "+group[0]+" : "+group[0].calculateMean(studentLabGrades));
        System.out.println("Moyenne de "+group[1]+" : "+group[1].calculateMean(studentLabGrades));
    }
}
```

Listing 4: Appel d'une méthode de calcul de moyenne.

2 TD Guidé numéro 2 : listes, garbage collector, attributs et méthodes statiques

2.1 Exercice 0 - Classe de base

- Dans un projet vierge, créez une classe Vehicule contenant trois attributs privés : une chaine de caractère `modele`, une vitesse sous forme d'un double `speed` et un double final `maxSpeed`.
- Codez un constructeur ayant la signature `public Vehicule(String modele, double maxSpeed);`(la vitesse initiale est nulle), ainsi qu'une méthode `public void accelerate(double acceleration);`. Evidemment, on ne peut pas aller au-delà de la vitesse maximale.
- Surchargez la méthode `toString` de manière à affficher les trois caractéristiques.
- Au sein d'un main codé dans une classe différente, testez votre classe.

- In a new project, create a 'Vehicule' class containing three private attributes: a string `modele`, a double `speed`, and a final double `maxSpeed`.
- Implement a constructor with the signature `public Vehicule(String modele, double maxSpeed);` (initial speed is zero), as well as a method `public void accelerate(double acceleration);`. Of course, the speed cannot exceed the maximum speed.
- Override the `toString` method to display the three attributes.
- In a `main` method coded in a different class, test your class.

2.1.1 Exercice 1 - Listes

- Dans le main, générez une ArrayList de Vehicule. Insérez les véhicules ci-dessous.
- Affichez la liste (on peut ne pas utiliser de boucle for). Testez
- Faites accélérez de 10 km/h chaque éléments de la liste. Testez
- Supprimer le troisième éléments de la liste. Testez
- Insérez en 2ième position un véhicule ("Pompier", 150 km/h). Testez

1	Fusée spatiale	30000
2	Voiture	130
3	Trotinette	8
4	Scooter	60
5	TGV	300

- In the main method, generate an 'ArrayList' of 'Vehicule'. Insert the vehicles listed below.
- Display the list (you may avoid using a for loop). Test.
- Accelerate each element in the list by 10 km/h. Test.
- Remove the third element from the list. Test.
- Insert a vehicle ("Firetruck", 150 km/h) in the second position. Test.

2.2 Exercice 2 - Attributs statiques

- Ajoutez à la classe "Vehicule" une référence vers un attribut privé statique "listOfVehicules". Cet attribut est une liste tous les véhicules existant.
- Modifiez le constructeur de manière à ce que la liste soit à jour.
- Ajoutez une méthode renvoyant un entier "indexInList" dont le but est de donner la position dans la liste du véhicule courant. La méthode indexOf des ArrayList fera cela très bien.
- Créez une méthode de destruction `void removeFromList(void);` qui retire le véhicule de la liste.
- Intégrez un getter pour l'attribut listOfVehicules.
- Fabriquez une autre classe de test et refaites l'exemple précédent, en utilisant la liste statique.

- Add a link to a private static attribute 'listOfVehicules' in the 'Vehicule' class. This attribute is a list of all existing vehicles.
- Also, add an integer attribute 'indexInList' to store the position of the current vehicle in the list (this attribute is not static).
- Modify the constructor to ensure the list is updated.
- Create a destruction method `void removeFromList(void);` that removes the vehicle from the list.
- Implement a getter for the 'listOfVehicules' attribute.
- Create another test class and repeat the previous example, using the static list.

2.2.1 Exercice 3 - Garbage collector

- Ajoutez une méthode statique permettant de connaître le nombre de Vehicule existant. Cette méthode regarde la taille de la liste.
- Surchargez la méthode de destructeur (la méthode finalize fera l'affaire, même si elle est obsolète à partir de Java 9) pour afficher un message `"Destruction de l'objet"+name`.
- Faites passez le garbage collector (Appel System.gc()) pour vérifier la supression de l'objet Trotinette.

- Add a static attribute to know the number of existing vehicles (you can refer to the videos).
- Override the destructor method (the ‘finalize‘ method will do, even though it is deprecated as of Java 9).
- Trigger the garbage collector (‘System.gc()‘ call) to check the deletion of the ‘Trotinette‘ object.

2.2.2 Exercice 4 - Méthodes statiques

- Créez une méthode statique getFastestVehicule qui renvoie le plus rapide des véhicules présent dans la liste (vitesse réelle, pas la vitesse maximale).
- Testez le résultat.

- Create a static method ‘getFastestVehicule‘ that returns the fastest vehicle in the list (actual speed, not maximum speed).
- Test the result.

3 TD Guidé numéro 3 : héritage, polymorphisme

Ce TD vise à illustrer l'héritage par le biais d'un exercice modélisant un système de fichier informatique. L'architecture UML est la suivante :
This session's aim is to show some of inheritance's property through an exercice modeling a computer file system. The UML diagram is given bellow :

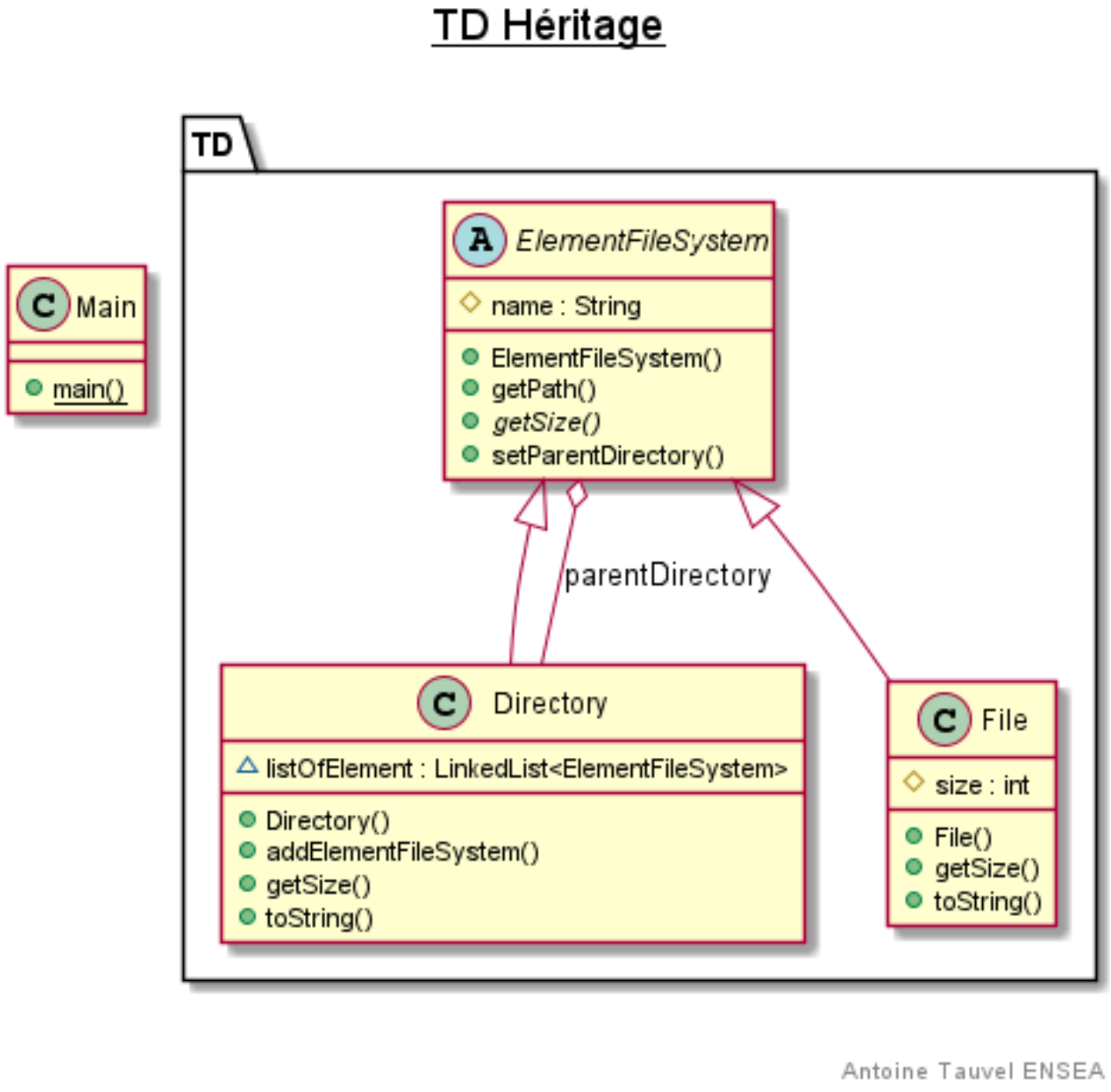

Figure 3.1: Diagramme de classe pour l'exercice sur l'héritage.

3.0.1 Exercice global - Classe abstraite

- Dans un projet vierge, créez une classe **abstraite** ElementFileSystem contenant deux attributs protégés : une chaine de caractère `name`, et une référence vers le dossier d'origine, qui sera lui même un Directory nommé ParentDirectory.
- Codez un constructeur ayant la signature `public ElementFileSystem(String name, Directory parentDirectory);`. Pour le moment, la classe Directory n'existe pas, on ne peux donc pas tester notre classe.
- Codez une méthode abstraite getSize renvoyant un entier.
- Codez un setter (non abstrait) pour l'élément ParentDirectory.

- In a new project, create an **abstract** class 'ElementFileSystem' containing two protected attributes: a string `name` and a reference to the parent directory, which will be a 'Directory' named 'parentDirectory'.
- Implement a constructor with the signature `public ElementFileSystem(String name, Directory parentDirectory);`. For now, the 'Directory' class does not exist, so we cannot test our class.
- Implement an abstract method 'getSize' that returns an integer.
- Implement a non-abstract setter for the 'parentDirectory' element.

3.1 Exercice global - File

- Créez la classe File qui hérite de la classe abstraite ElementFileSystem. Une erreur apparaît. Pourquoi ?
- Ajoutez à la classe un élément protégé `size` donnant la taille du fichier en octet.
- Corrigez l'erreur en surchargeant la méthode getSize.
- Codez un constructeur ayant la signature `public File(String name, Directory parentDirectory, int size);`.

- Create the 'File' class that inherits from the abstract class 'ElementFileSystem'. An error appears. Why?
- Add a protected element `size` to the class, representing the file size in bytes.
- Fix the error by overriding the 'getSize' method.
- Implement a constructor with the signature `public File(String name, Directory parentDirectory, int size);`.

3.2 Exercice global - Directory

- Créez la classe Directory qui hérite de la classe abstraite ElementFileSystem.
- Ajoutez à la classe un élément protégé `LinkedList<ElementFileSystem> listOfElements` qui modélise la liste des fichiers et des répertoires dans le répertoire.
- Codez un constructeur ayant la signature `public Directory(String name, Directory parentDirectory);`, qui fait juste appel au constructeur de la superclasse et initialise la liste.
- Codez la méthode `public void addElementFileSystem(ElementFileSystem e)` qui ajoute un élément dans la liste. Appelez cette méthode dans le constructeur de ElementFileSystem.
- Surchargez la méthode getSize(). A l'aide d'une boucle for, elle fait appel à la méthode getSize de tous les éléments de la liste (répertoire et fichiers indifféremment).
- A l'aide du code ci-dessous, vérifiez le fonctionnement de vos trois classes. Vous devez obtenir comme taille respectivement 200, 1450, 450 et 1000.

- Create the ‘Directory‘ class that inherits from the abstract class ‘ElementFileSystem‘.
- Add a protected element `LinkedList<ElementFileSystem> listOfElements` to the class, which models the list of files and directories within the directory.
- Implement a constructor with the signature `public Directory(String name, Directory parentDirectory);`, which simply calls the superclass constructor and initializes the list.
- Implement the method `public void addElementFileSystem(ElementFileSystem e)` that adds an element to the list. Call this method in the constructor of ‘ElementFileSystem‘.
- Override the ‘getSize()‘ method. Using a ‘for‘ loop, it should call the ‘getSize‘ method of all elements in the list (both directories and files).
- Using the code below, verify the functionality of your three classes. You should obtain sizes of 200, 1450, 450, and 1000, respectively.

```java
public class Main {

    public static void main(String[] args) {
        // write your code here
        Directory d1 = new Directory("",null);
        Directory d2 = new Directory ("MyDir",d1);
        Directory d3 = new Directory ("MyOtherDir",d1);

        File f1 = new File("exemple1.txt",d2,200);
        File f2 = new File("exemple2.txt",d2,250);
        File f3 = new File("exemple3.txt",d3,1000);

        System.out.println(f1.getSize());
        System.out.println(d1.getSize());
        System.out.println(d2.getSize());
        System.out.println(d3.getSize());
    }
}
```

Listing 5: Code de test de nos classes Files

Alerte polymorphisme ! Même s’il ne fait pas partie de cette séquence, nous avons utilisé ici un polymorphisme sur la méthode getSize(). Vous voyez ici se déployer la puissance de ce concept.
Polymorphism Alert! *Although it is not part of this sequence, we have used polymorphism on the ‘getSize()‘ method here. You can see the power of this concept being demonstrated.*

3.3 On rafine les méthodes - getPath()

On souhaite rajouter une méthode getPath() renvoyant une chaîne de caractère qui indique la position de l'élément. Dans quelle classe doit-elle se trouver ?
Codez cette méthode. On part d'une chaîne de caractère vide, et d'une référence vers l'objet en cours, et on remonte chaque élément vers le parent tant que le résultat n'est pas `null`.
Testez cette méthode.

We want to add a method 'getPath()' that returns a string indicating the position of the element. In which class should it be located?
Implement this method. Start with an empty string and a reference to the current object, and traverse each element up to the parent as long as the result is not `null`.
Test this method.

3.4 On rafine les méthodes - toString()

On souhaite surcharger toString de manière à afficher juste le nom du fichier et sa taille si c'est un fichier, la liste de tous les fichiers et sous dossiers si c'est un répertoire.
Codez et testez cette méthode.

We want to override 'toString' to display just the file name and its size if it is a file, or the list of all files and subdirectories if it is a directory.
Implement and test this method.

3.5 Solution

En cas de soucis, vous trouverez la solution à ces exercices à l'adresse suivante :
You may find the solution to this small problem here :
`https://github.com/antoineTauvelENSEA/ElementFileSystem/tree/master/src`

3.6 Bonus

Codez une méthode statique `ElementFileSystem findByName (String name, Directory directory);` qui renvoie le premier objet ayant exactement ce nom.
Codez une méthode statique `File findByWeight(Directory directory);` qui renvoie le fichier le plus lourd au sein d'un répertoire donnée (première version : sans les sous-répertoire, puis avec les sous répertoire inclus).

Implement a static method `ElementFileSystem findByName(String name, Directory directory);` that returns the first object with exactly that name.
Implement a static method `File findByWeight(Directory directory);` that returns the heaviest file within a given directory (first version: without subdirectories, then with subdirectories included).

4 Projet : un dongeon crawler de base 4h.

Le but de l'ensemble du TP est de programmer la base d'un petit jeu permettant d'afficher un dongeon de manière statique (sans caméra) et des sprites pouvant se mouvoir dans ce dongeon en respectant une physique élémentaire (gestion des collisions au minimum).
Nous vous proposons quatre niveau de difficulté pour cette partie, qui est très serré en terme d'horaire. Vous ne serez pas jugé sur le niveau de difficulté mais sur votre implication sur la partie suivante ! N'ayez donc pas honte à regarder la solution lorsque c'est nécessaire.
The goal of the entire lab exercise is to program the basis of a small game that displays a dungeon statically (without a camera) and sprites that can move within this dungeon while adhering to basic physics (minimum collision management).
We offer you four levels of difficulty for this part, which is very tight in terms of time.
You will not be judged on the level of difficulty but on your involvement in the next part! So don't hesitate to look at the solution when necessary.

Difficulté	Consigne
Krilin	Allez chercher en ligne la solution à l'adresse : `https://github.com/antoineTauvelENSEA/FISE_2024_2025_Dungeon_Crawler`. Commentez l'ensemble du code en le faisant fonctionner. *You can find a complete solution on the adress above*
Goku	Suivez l'ensemble du tutoriel ci dessous. *Follow the complete tutorial*
Saiyan	Suivez l'ensemble du tutoriel mais sans reprendre les codes préécrit. Retrouvez ces fonctionnalités plus difficiles. *Follow the complete tutorial without the pre-written code.*
Super Saiyan	Ne suivez que le premier chapitre permettant de récupérer les assets. A l'aide du diagramme de classe présent dans le second chapitre, reconstituez l'ensemble du code.*Just get the assets. Using the UML class diagram, write the complete code.*

4.1 Mise en place du projet et récupération des assets

- Générez un nouveau projet en Java.
- Générez une classe Main affichant un message dans la console pour tester votre configuration de fonctionnement.
- En vous rendant à l'url suivante : `https://github.com/antoineTauvelENSEA/FISE_2024_2025_Dungeon_Crawler`, récuperer le contenu des répertoires /img (contenant les images utiles au jeu) et /data (contenant un niveau de test).

- Generate a new Java project.
- Create a 'Main' class that displays a message in the console to test your setup.
- Visit the following URL: `https://github.com/antoineTauvelENSEA/FISE_2024_2025_Dungeon_Crawler` to retrieve the contents of the /img directory (containing the images needed for the game) and the /data directory (containing a test level).

4.2 Conception UML

Cette conception s'articule autour de trois moteurs. Chacun de ces moteurs s'éxécute dans un Thread spécifique cadencé par un Timer (et oui, ici aussi il y a des timers). On éxécute ces Timers toutes les 50 millisecondes.
Même si l'étude des Threads relève du cours de Java avancé, nous allons ici nous limiter au minimum.
Voici le rôle de chacun des moteurs :

- Le RenderEngine permet d'afficher tous les éléments graphiques. Certains sont statiques et d'autres sont animés.
- Le PhysicEngine est chargé de modéliser la physique du jeu. Pour nous c'est relativement simple : on se contente de gérer une vitesse constante et des collisions.
- Le GameEngine ne gère ici que les interractions avec l'utilisateur (en utilisant le clavier). Dans un projet plus ambitieux, il permettrait aussi de charger les niveaux, de gérer les game over etc[1].

Pour être certains que ces trois moteurs soient tous instanciables au sein d'un Timer, ils doivent implémenter une méthode update. On définit donc une interface Update.
Logiquement, la classe principal a un exemplaire de chaque moteur.
This design revolves around three engines. Each of these engines runs in a specific thread timed by a timer (yes, there are timers here too). These timers are executed every 50 milliseconds.
Even though the study of threads falls under the advanced Java course, we will limit ourselves to the essentials here.
Here is the role of each engine:

- *The 'RenderEngine' is responsible for displaying all graphical elements. Some are static while others are animated.*
- *The 'PhysicsEngine' is tasked with modeling the game's physics. For us, this is relatively simple: we only manage constant speed and collisions.*
- *The 'GameEngine' handles user interactions (using the keyboard). In a more ambitious project, it would also load levels, manage game overs, etc.*[2]

To ensure that all three engines can be instantiated within a timer, they must implement an 'update' method. We thus define an 'Update' interface.
Logically, the main class has an instance of each engine.

[1] Il est souvent codé sous la forme d'une grosse machine à état. Vu le temps dont nous disposons, il est simplifié pour ne gérer que les interactions claviers.
[2] It is often coded as a large state machine. Given the time we have, it is simplified to only manage keyboard interactions.

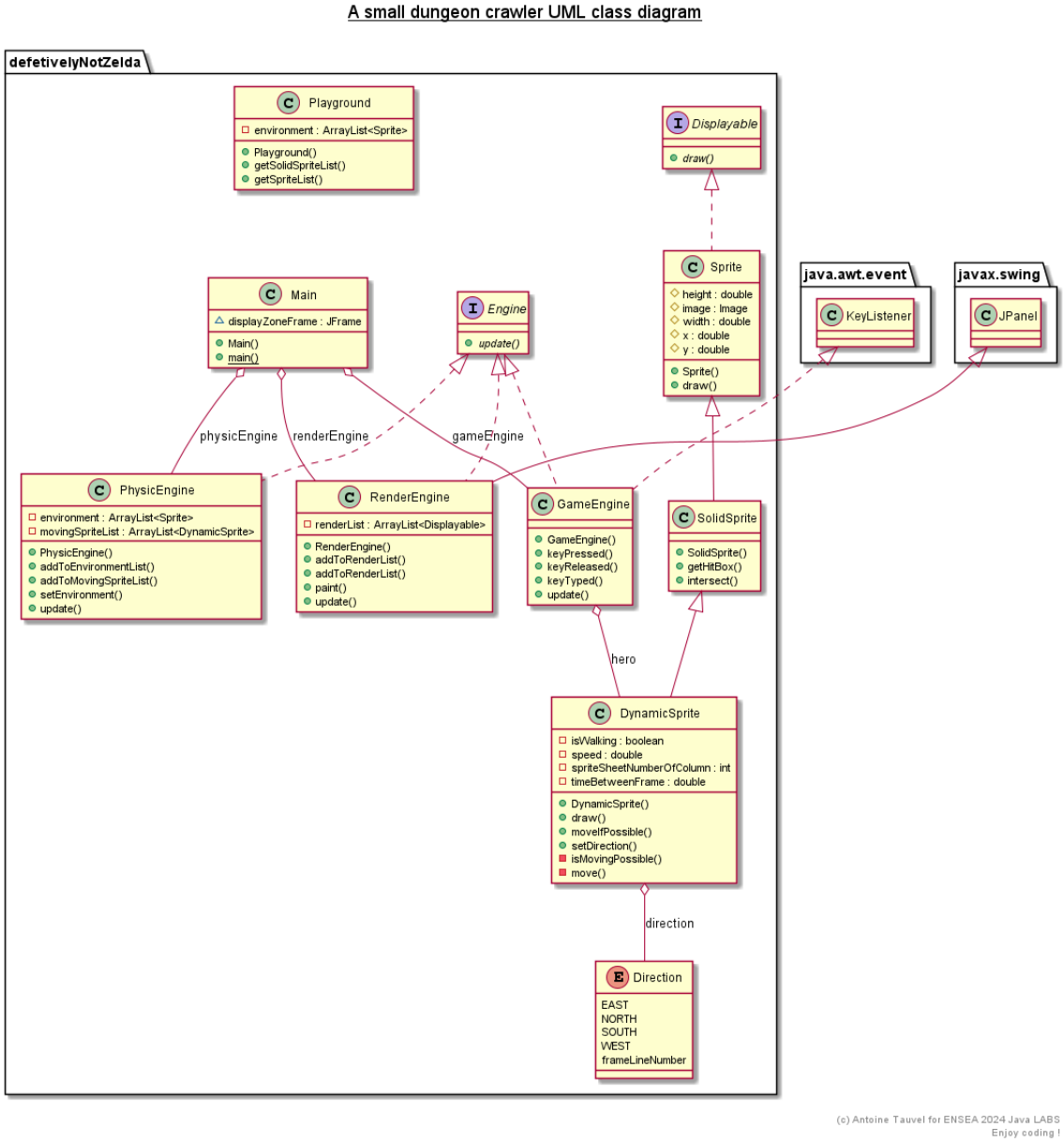

Figure 4.1: Le diagramme UML du projet complet

Nous avons ensuite une cascade de classe héritant d'une interface Displayable. Cette interface implémente une seule méthode abstraite : draw, qui devra définir la méthode d'affichage de chaque nature de classe.

- La classe Sprite permet de modéliser des éléments de décor sans substance physique. Le moteur de rendu utilise donc une liste de Sprite.
- La classe SolidSprite permet de modéliser des éléments de décor ayant une hitbox, comme les arbres ou les rochers. Le moteur physique emploi donc une liste de SolidSprite.
- La classe DynamicSprite permet de modélier des éléments animé. Dans notre version simplifié, il n'y a que le héro. Le moteur de jeu a une référence vers le héro.

Enfin, pour terminer cette présentation, un sprite dynamique doit forcément regarder dans une direction, et éventuellement s'y rendre. C'est le rôle de la classe énuméré Direction. Les quatre valeurs (NORTH, SOUTH, EAST, WEST) sont associées à quatre entiers (2,0,3,1) qui correspond au numéro de la ligne correspondant au dessin du héro dans la spritesheet.

C'est ici la fin des informations pour le codage en mode Super Sayian. Vous avez normalement assez d'information pour reconstituer le code !

We then have a cascade of classes inheriting from a 'Displayable' interface. This interface implements a single abstract method: 'draw', which must define the display method for each type of class.

- *The 'Sprite' class models decorative elements without physical substance. The render engine thus uses a list of 'Sprite'.*
- *The 'SolidSprite' class models decorative elements with a hitbox, such as trees or rocks. The physics engine uses a list of 'SolidSprite'.*
- *The 'DynamicSprite' class models animated elements. In our simplified version, there is only the hero. The game engine has a reference to the hero.*

Finally, to conclude this presentation, a dynamic sprite must always face a direction and potentially move in that direction. This is the role of the 'Direction' enumeration class. The four values (NORTH, SOUTH, EAST, WEST) are associated with four integers (2, 0, 3, 1) that correspond to the row number for the hero's drawing in the sprite sheet.

This is the end of the information for coding in Super Saiyan mode. You should have enough information to reconstruct the code!

4.3 La classe Main

4.3.1 Usage de Swing

Comme vous l'avez vu dans les vidéos de cours, le framework swing permet de créer des GUI[3] facilement. Quoique très ancien (2000), il est toujours d'actualité pour un résultat rapide. Son principal concurrent, qui devait être son successeur est le framework JavaFx mais il nécessite de gérer correctement les dépendances de paquet.
Swing affiche les éléments dans une fenêtre graphique JFrame devant elle même comprendre un ou plusieurs JPanel.
Nous allons donc utiliser un JFrame pour obtenir un affichage d'une fenêtre. Nous allons utiliser trois méthodes : setSize fixe la taille en pixel, setDefaultCloseOperation spécifie ce qu'il doit se passer lors de l'appui sur la croix de fermeture de l'application (dans notre cas, la fin du programme), et setVisible, qui déclenche l'affichage.

Codez le constructeur de la clase Main de manière à afficher une fenêtre de taille 400 par 600 pixels. Vous pouvez vous aider si nécessaire du code ci-dessous.

```
1  public class Main{
2
3      JFrame displayZoneFrame;
4
5      public Main() throws Exception{
6          displayZoneFrame = new JFrame("Java Labs");
7          displayZoneFrame.setSize(400,600);
8          displayZoneFrame.setDefaultCloseOperation(EXIT_ON_CLOSE);
9          displayZoneFrame.setVisible(true);
10     }
```

[3] Graphical User Interface

```
11
12      public static void main (String[] args) throws Exception {
13          Main main = new Main();
14      }
15  }
```

Listing 6: La base de notre projet

As you saw in the course videos, the Swing framework allows for easy creation of GUIs[4]. Although it is quite old (2000), it is still relevant for quick results. Its main competitor, which was supposed to be its successor, is the JavaFX framework, but it requires proper management of package dependencies. Swing displays elements in a graphical window ('JFrame'), which must itself contain one or more 'JPanel'.
We will therefore use a 'JFrame' to display a window. We will use three methods: 'setSize' sets the size in pixels, 'setDefaultCloseOperation' specifies what should happen when the close button of the application is pressed (in our case, the end of the program), and 'setVisible' triggers the display.

Code the constructor of the 'Main' class to display a window sized 400 by 600 pixels. You may use the code above for assistance if needed.

4.3.2 L'interface Engine

Codez l'interface Engine comportant une unique méthode `void update();`.

- Peut-on instancier une interface ?
- A quelle condition une classe peut elle implémenter l'interface Engine ?
- Une classe peut-elle implémenter plusieurs interfaces ?

Code the 'Engine' interface containing a single method `void update();`.

- Can an interface be instantiated?
- Under what condition can a class implement the 'Engine' interface?
- Can a class implement multiple interfaces?

Nous allons maintenant créer le squelette de la classe RenderEngine.

Codez le squelette de RenderEngine, implémentant l'interface Engine.
Affichez un message dans la console lors de l'éxécution de la méthode "update".
Une fois le bon fonctionnement validé, vous pouvez retirer ce code d'affichage.
Au sein de la classe Main, utiliser le codes ci-dessous pour instancier la classe RenderEngine au sein d'une instance de la classe Timer (une classe du framework Swing).
Vérifiez le bon fonctionnement. J'attire votre attention sur la ligne 14 de ce code, qui utilise une *lambda expression* pour définir dynamiquement le contenu d'une fonction. Nous étudierons ce concept plus avant lors du cours de Java avancé.

[4] Graphical User Interface

```java
public class Main{

    JFrame displayZoneFrame;

    RenderEngine renderEngine;

    public Main() throws Exception{
        displayZoneFrame = new JFrame("Java Labs");
        displayZoneFrame.setSize(400,600);
        displayZoneFrame.setDefaultCloseOperation(EXIT_ON_CLOSE);

        renderEngine = new RenderEngine();

        Timer renderTimer = new Timer(50,(time)-> renderEngine.update());

        renderTimer.start();
        displayZoneFrame.setVisible(true);
    }

    public static void main (String[] args) throws Exception {
        Main main = new Main();
    }
}
```

Listing 7: Code d'affichage d'une fenêtre en Swing

We will now create the skeleton of the 'RenderEngine' class.

Code the skeleton of 'RenderEngine', implementing the 'Engine' interface.
Display a message in the console when the "update" method is executed.
Once the functionality is validated, you can remove this display code.
Within the 'Main' class, use the code below to instantiate the 'RenderEngine' class within an instance of the 'Timer' class (a class from the Swing framework). Verify its proper functioning. I draw your attention to line 14 of this code, which uses a *lambda expression* to dynamically define the contents of a function. We will study this concept further in the advanced Java course.

4.4 Render Engine

Comme indiqué plus haut, le framework swing affiche des éléments sur un panneau. Il doit synchroniser l'affichage, aussi la classe RenderEngine doit elle hériter de la classe JPanel (sans quoi l'affichage n'est pas synchronisé).

- Faites hériter la classe GameRender de JPanel.
- Ajoutez dans le constructeur de la classe Main une ligne permettant d'additionner notre instance de GameRender aux élément affichables, par exemple : `displayZoneFrame.getContentPane().add(renderEngine);`

As mentioned earlier, the Swing framework displays elements on a panel. It must synchronize the

display; therefore, the 'RenderEngine' class must inherit from the 'JPanel' class (otherwise, the display will not be synchronized).

- Have the 'GameRender' class inherit from 'JPanel'.
- Add a line in the constructor of the 'Main' class to add our instance of 'GameRender' to the displayable elements, for example: `displayZoneFrame.getContentPane().add(renderEngine);`

4.4.1 La classe Sprite

Pour mémoire, cette classe permet de modéliser n'importe quel élément affichable.

- Fabriquez une interface Displayable avec une unique méthode `public void draw(Graphics g);`.
- Créez une classe Sprite implémentant l'interface Displayable. Cette classe a comme attribues privés :
  - Une image.
  - Une position, modélisée par deux *double* nommées x et y.
  - Une taille, modélisée par deux *double* nommées width et height.
- Créez un constructeur prenant en compte les cinq attribues.
- Surchargez la méthode `public public void draw(Graphics g)` de manière à afficher l'image en x et y. Cela se fait en un appel de méthode sur le graphique : g.drawImage (à vous de trouver les bons paramètres).

For reference, this class allows modeling any displayable element.

- Create a 'Displayable' interface with a single method `public void draw(Graphics g);`.
- Create a 'Sprite' class implementing the 'Displayable' interface. This class has private attributes:
  - An image.
  - A position, modeled by two *double* values named x and y.
  - A size, modeled by two *double* values named width and height.
- Create a constructor that takes into account the five attributes.
- Override the method `public void draw(Graphics g)` to display the image at x and y. This is done with a method call on the graphics: g.drawImage (you need to find the right parameters).

4.4.2 La classe RenderEngine

La classe GameRender fait dans l'état actuel du code un appel régulier à la méthode update. Nous allons y placer maintenant les méthodes et attributs permettant l'affichage.

- Ajouter l'attribut privé `renderList` qui est une liste de Displayable.
- Ecrivez un constructeur par défaut qui initialise la liste `renderList`
- Créez un setter pour renderList.
- Créez une méthode `public void addToRenderList(Displayable displayable)` permettant d'ajouter un élément à la renderList.

Nous allons maintenant mettre en oeuvre un point clé du programme : le polymorphisme. En effet, dans la prochaine méthode, même si pour le moment, on ne va traiter qu'un seul Sprite, la "bonne" méthode `draw` sera appelé quelque soit la nature réel de l'objet (Sprite ou SolidSprite ou DynamicSprite).

Surchargez la méthode `public void paint(Graphics g)`. Cette méthode commence par appeler la méthode paint de la super-classe (Classe mère JPanel), puis, pour chaque élément de la renderList, appelle la méthode draw.
Il ne reste plus qu'à "peindre" le composant régulièrement. Pour cela, il suffit d'appeler la méthode `repaint()` au sein de la méthode update (pas paint, repaint génère le paramètre graphique).
Testez l'ensemble à l'aide des lignes ci-dessous. Si le résultat n'affiche pas un arbre, merci de corriger les problèmes en faisant ci nécessaire appel à l'enseignant.

```
Sprite test = new Sprite(200,300,
            ImageIO.read(new File("./img/tree.png")),64,64);
renderEngine.addToRenderList(test);
```

Fin théorique de la séance 1. Merci de vous remettre à jour avant le début de la séance 2.
The 'GameRender' class currently makes a regular call to the 'update' method. We will now place the methods and attributes necessary for display.

- Add the private attribute `renderList`, which is a list of 'Displayable'.
- Write a default constructor that initializes the `renderList`.
- Create a setter for `renderList`.
- Create a method `public void addToRenderList(Displayable displayable)` that allows adding an element to the `renderList`.

We will now implement a key point of the program: polymorphism. Indeed, in the next method, even though we will only process a single 'Sprite' for now, the "correct" 'draw' method will be called regardless of the actual nature of the object (whether it's a 'Sprite', 'SolidSprite', or 'DynamicSprite').

Override the method `public void paint(Graphics g)`. This method starts by calling the paint method of the superclass (the parent class 'JPanel'), then for each element in the `renderList`, it calls the 'draw' method.
All that remains is to "paint" the component regularly. To do this, simply call the `repaint()` method within the `update` method (not paint; repaint generates the graphic parameter).
Test the whole setup using the lines below. If the result does not display a tree, please correct the issues, making sure to consult the teacher if necessary.

You must be here at the end of session 1. If needed, use the online solution to speed up your work.

4.4.3 La classe SolidSprite

Créez la classe SolidSprite qui hérite de la classe Sprite.
Cette classe a pour le moment une seule méthode : un constructeur équivalent à celui de Sprite (appel à la méthode `super`)

Create the class 'SolidSprite' that inherits from the 'Sprite' class.
This class currently has only one method: a constructor equivalent to that of 'Sprite' (calling the `super` method).

4.4.4 L'énuméré Direction

Créez une classe énumérée nommée Direction pouvant prendre comme valeur NORTH, SOUTH, EAST, WEST. Cette classe permet de récupérer une valeur entière associé de manière unique à l'énuméré. Vous pouvez reprendre le code ci-dessous, ou essayer de l'écrire vous-même...

Create an enum class named 'Direction' that can take the values 'NORTH', 'SOUTH', 'EAST', and 'WEST'. This class allows you to retrieve a unique integer value associated with each enumerated constant. You can use the code below, or try to write it yourself...

```
public enum Direction {
    NORTH(2),SOUTH(0),EAST(3),WEST(1);
    private int frameLineNumber;

    Direction(int frameLineNumber) {
        this.frameLineNumber = frameLineNumber;
    }

    public int getFrameLineNumber() {
        return frameLineNumber;
    }
}
```

Listing 8: Code de la classe énumérée Direction

4.4.5 La classe DynamicSprite et l'affichage d'une animation

Créez la classe DynamicSprite qui hérite de la classe SolidSprite.
Cette classe comporte les attributs supplémentaires suivants :

- Un booléen nommé `isWalking` valant `true` par défaut.
- Un `double` nommée speed valant 5 par défaut.
- Un attribut final de type int : spriteSheetNumberOfColumn (valant 10 par défaut).
- Un attribut de type int nommé `timeBetweenFrame` valant par défaut 200 (200 millisecondes entre deux images).
- Un attribu de type `Direction` nommé direction.

Codez un setter pour la variable direction.

Create the class 'DynamicSprite' that inherits from the 'SolidSprite' class.
This class contains the following additional attributes:

- A boolean named `isWalking` set to `true` by default.
- A `double` named `speed` set to 5 by default.
- A final attribute of type int: `spriteSheetNumberOfColumn` (set to 10 by default).
- An int attribute named `timeBetweenFrame` set to 200 by default (200 milliseconds between two frames).
- An attribute of type `Direction` named `direction`.

Code a setter for the variable `direction`.

Si ce n'est pas déjà fait, regarder la spritesheet du héros ci-dessous.

Figure 4.2: La spritesheet du héros.

Elle fait 480 par 202 pixels et comporte 10 images par 4, chaque image faisant 48 pixels par 51.
Le but est, une fois cette spritesheet chargée en mémoire, de n'en afficher que la portion désirée. Il nous faut donc surcharger la méthode draw.

Surchargez la méthode draw, qui intègre les comportements suivant :

- On calcul un entier index donnant le numéro de l'image à afficher. Pour cela, on divise le temps courant (`System.currentTimeMillis()`) par le temps entre deux trames modulo le nombre total de trame.
- On récupère un entier attitude correspondant à la valeur numérique de la direction.
- On affiche l'image avec drawImage et les bons paramètres. La source démarre à $(index * width, attitude * height)$ et finit à $((index + 1) * width, (attitude + 1 * height))$
- Intégrer un héro dans le constructeur de la classse Main et vérifier son animation, comme indiqué ci-dessous.

```
DynamicSprite hero = new DynamicSprite(200,300,
            ImageIO.read(new File("./img/heroTileSheetLowRes.png")),48,50);
renderEngine.addToRenderList(hero);
```

Override the draw method, which includes the following behaviors:

- Calculate an integer `index` representing the number of the image to display. To do this, divide the current time (`System.currentTimeMillis()`) by the time between two frames modulo the total number of frames.
- Retrieve an integer `attitude` corresponding to the numerical value of the direction.
- Display the image using `drawImage` with the correct parameters. The source starts at $(index * width, attitude * height)$ and ends at $((index + 1) * width, (attitude + 1) * height)$.
- Integrate a hero in the constructor of the `Main` class and verify its animation, as indicated below.

4.5 Game Engine

On souffle un peu avec la classe la plus simple ! Courage, on y est presque !

- Créez une classe GameEngine qui implémente les interfaces Engine et KeyListener.
- Créez en attribut une référence privée finale vers le héros.
- Créez un constructeur qui fixe la référence vers le héros.
- Au sein de la méthode keyPressed, faites un switch sur la variable e.getKeyCode de manière à positionner la variable direction du Héros. Pour information, le code des flèches est sur le format : KeyEvent.VK_UP pour la flèche vers le haut.
- Rajouter dans le main la création d'une instance de GameEngine.
- En vous inspirant du code d'instanciation du RenderEngine, implémentez une instance de Timer exécutant le GameEngine dans le constructeur du Main (facultatif, cf remarque ci-dessous).
- Dans le constructeur du main, ajouter un keyListener de la manière suivante : `displayZoneFrame.addKeyListener(gameEngine);`
- Testez votre code : le héros doit avoir une animation dans la direction de la flèche indiqué par le clavier, sans mouvement.

Vous serez certainement surpris que la méthode update reste vide. C'est lié à la structure simplifié de notre projet, compte tenu du temps disponible. Un Game Engine bien conçu intègre une FSM (machine à état) qui doit évoluer à intervalle périodique.

Let's take a break with the simplest class! Hang in there, we are almost there!

- Create a class `GameEngine` that implements the `Engine` and `KeyListener` interfaces.
- Create a private final reference attribute for the hero.
- Create a constructor that sets the reference to the hero.
- Within the `keyPressed` method, use a switch statement on the variable `e.getKeyCode` to set the hero's direction variable. For your information, the key codes for the arrows are in the format: `KeyEvent.VK_UP` for the up arrow.
- Add the creation of an instance of `GameEngine` in the main method.
- Inspired by the instantiation code of the `RenderEngine`, implement a `Timer` instance executing the `GameEngine` in the `Main` constructor (optional, see remark below).
- In the `Main` constructor, add a key listener as follows: `displayZoneFrame.addKeyListener(gameEngine);`
- Test your code: the hero should have an animation in the direction of the arrow indicated by the keyboard, without any movement.

You will probably be surprised that the `update` method remains empty. This is related to the

simplified structure of our project, given the available time. A well-designed game engine incorporates a FSM (finite state machine) that should evolve at regular intervals.

4.6 Physic Engine

4.6.1 Modification de la classe DynamicSprite

On doit maintenant écrire la méthode de déplacement de nos éléments dynamiques. Pour cela, on se contente d'addtionner la vitesse à la position à intervalle régulier.
We must now write the moving method of our dynamic sprite. We do that by adding the speed at the position every other delay.
Par exemple :

```
switch (direction){
            case NORTH -> {
                this.y-=speed;
            }
```

Mais il faut avant vérifier que le mouvement est possible. Nous allons procéder en deux temps :

- D'abord, on calcul une hitbox décalé du mouvement prévu.
- Ensuite, on recherche toutes les collisions avec l'ensemble des éléments solides.

La classe Rectangle2D, qui fait parti de Swing, intègre une méthode intersect qui renvoi un booléen...

- Ecrivez une méthode privé au sein de la classe DynamicSprite `private void move()` qui déplace le Sprite.
- Ecrivez une méthode privé `private boolean isMovingPossible(ArrayList<Sprite> environment)` qui créé une variable local `hitBox` de type `Rectangle2D.Double` correspondant au sprite déplacé de Speed pixel dans la bonne direction.
- Complétez la méthode `isMovingPossible` à l'aide d'une boucle for qui itère tous les éléments du paramètre environnement. Si l'élément est un SolidSprite et qu'il y a une intersection, et que l'élément n'est pas le DynamicSprite lui même, alors on renvoie `false`.
- A la fin de la boucle for, on renvoie true.
- Enfin, codez la fonction `public void moveIfPossible(ArrayList<Sprite> environment)`, qui appelle intelligemment les deux fonctions précédentes.

Deadline Panic ! Si vous n'avez pas le temps, vous pouvez vous contentez de coder la fonction move et passer à la suite. Le héros passera alors au travers des éléments de décors.

But first, we need to check if the movement is possible. We will proceed in two steps:

- First, we calculate a hitbox offset by the intended movement.
- Then, we check for all collisions with all solid elements.

The class `Rectangle2D`, which is part of Swing, includes an `intersect` method that returns a boolean...

- Write a private method within the `DynamicSprite` class `private void move()` that moves the Sprite.
- Write a private method `private boolean isMovingPossible(ArrayList<Sprite> environment)` that creates a local variable `hitBox` of type `Rectangle2D.Double` corresponding to the sprite moved by Speed pixels in the correct direction.
- Complete the `isMovingPossible` method using a for loop that iterates through all elements of the environment parameter. If the element is a `SolidSprite` and there is an intersection, and the element is not the `DynamicSprite` itself, then return `false`.
- At the end of the for loop, return true.
- Finally, code the function `public void moveIfPossible(ArrayList<Sprite> environment)`, which intelligently calls the two previous functions.

Deadline Panic! If you don't have time, you can just code the `move` function and move on. The hero will then pass through the decorative elements.

4.6.2 La classe PhysicEngine

- Creez une classe PhysicEngine, qui implémente l'interface Engine.
- Cette classe a deux attributs :
  - Une liste movingSpriteList qui contient l'ensemble des Sprites à mouvoir
  - Une liste environment qui contient l'ensemble des sprites solides.
- Créez une méthode pour ajouter un élement à la movingSpriteList.
- Créez un setter pour la liste d'environnement.
- Enfin, surchargez la méthode update avec une boucle for qui appelle la méthode moveIfPossible pour tous les éléments de la movingSpriteList
- Testez à l'aide du Main ci dessous. Le personnage devrait se mouvoir jusqu'à "se cogner" contre le rocher.
- *Create a class PhysicEngine that implements the Engine interface.*
- *This class has two attributes:*
  - *A list movingSpriteList that contains all the Sprites to be moved.*
  - *A list environment that contains all the solid sprites.*
- *Create a method to add an element to the movingSpriteList.*
- *Create a setter for the environment list.*
- *Finally, override the update method with a for loop that calls the moveIfPossible method for all elements in the movingSpriteList.*

- *Test using the Main code below. The character should move until it "bumps" against the rock.*

```
public Main() throws Exception{
        displayZoneFrame = new JFrame("Java Labs");
        displayZoneFrame.setSize(400,600);
        displayZoneFrame.setDefaultCloseOperation(EXIT_ON_CLOSE);

        DynamicSprite hero = new DynamicSprite(0,300,
                ImageIO.read(new File("./img/heroTileSheetLowRes.png")),48,50);

        renderEngine = new RenderEngine();
        physicEngine = new PhysicEngine();
        gameEngine = new GameEngine(hero);

        Timer renderTimer = new Timer(50,(time)-> renderEngine.update());
        Timer gameTimer = new Timer(50,(time)-> gameEngine.update());
        Timer physicTimer = new Timer(50,(time)-> physicEngine.update());

        renderTimer.start();
        gameTimer.start();
        physicTimer.start();

        displayZoneFrame.getContentPane().add(renderEngine);
        displayZoneFrame.setVisible(true);

      SolidSprite testSprite = new SolidSprite(250,300,(new File("./img/rock.png")),64,64);
        renderEngine.addToRenderList(testSprite);
        renderEngine.addToRenderList(hero);
        physicEngine.addToMovingSpriteList(hero);
        physicEngine.setEnvironment(new ArrayList<SolidSprite>(testSprite));
}
```

Listing 9: Code de test du moteur physique

4.7 Level Loader : la classe PlayGround

Pour des raisons de temps, je vous fournis le code de la classe PlayGround, il n'y a plus qu'à copier / coller...
We don't have enough time, so I give you the PlayGround class. Copy and Paste it in your project.

```
public class Playground {
    private ArrayList<Sprite> environment = new ArrayList<>();

    public Playground (String pathName){
        try{
            final Image imageTree = ImageIO.read(new File("./img/tree.png"));
            final Image imageGrass = ImageIO.read(new File("./img/grass.png"));
            final Image imageRock = ImageIO.read(new File("./img/rock.png"));
            final Image imageTrap = ImageIO.read(new File("./img/trap.png"));

            final int imageTreeWidth = imageTree.getWidth(null);
```

```
        final int imageTreeHeight = imageTree.getHeight(null);

        final int imageGrassWidth = imageGrass.getWidth(null);
        final int imageGrassHeight = imageGrass.getHeight(null);

        final int imageRockWidth = imageRock.getWidth(null);
        final int imageRockHeight = imageRock.getHeight(null);

        BufferedReader bufferedReader = new BufferedReader(new FileReader(pathName));
        String line=bufferedReader.readLine();
        int lineNumber = 0;
        int columnNumber = 0;
        while (line!= null){
            for (byte element : line.getBytes(StandardCharsets.UTF_8)){
                switch (element){
                    case 'T' : environment.add(new SolidSprite(columnNumber*imageTreeWidth,
                            lineNumber*imageTreeHeight,imageTree, imageTreeWidth, imageTreeHeight));
                                break;
                    case ' ' : environment.add(new Sprite(columnNumber*imageGrassWidth,
                            lineNumber*imageGrassHeight, imageGrass, imageGrassWidth, imageGrassHeight));
                        break;
                    case 'R' : environment.add(new SolidSprite(columnNumber*imageRockWidth,
                            lineNumber*imageRockHeight, imageRock, imageRockWidth, imageRockHeight));
                        break;
                }
                columnNumber++;
            }
            columnNumber =0;
            lineNumber++;
            line=bufferedReader.readLine();
        }
    }
    catch (Exception e){
        e.printStackTrace();
    }
}

public ArrayList<Sprite> getSolidSpriteList(){
    ArrayList <Sprite> solidSpriteArrayList = new ArrayList<>();
    for (Sprite sprite : environment){
        if (sprite instanceof SolidSprite) solidSpriteArrayList.add(sprite);
    }
    return solidSpriteArrayList;
}

public ArrayList<Displayable> getSpriteList(){
    ArrayList <Displayable> displayableArrayList = new ArrayList<>();
    for (Sprite sprite : environment){
        displayableArrayList.add((Displayable) sprite);
    }
    return displayableArrayList;
}
```

```
}
```

Listing 10: Code de la classe PlayGround permettant de charger le niveau.

4.8 Le main final

Il n'y a plus qu'à intégrer une instance de PlayGround dans le constructeur du Main, et à intégrer ses listes de Sprite et SolidSprite dans les renderList et environment du RenderEngine et PhysicEngine.
Félicitations, vous avez maintenant une base saine pour la suite !

All that's left is to integrate an instance of PlayGround in the Main constructor and to integrate its lists of Sprite and SolidSprite into the renderList and environment of the RenderEngine and PhysicEngine.
Congratulations, you now have a solid foundation for what comes next!

Figure 4.3: Capture d'écran du rendu à ce stade

5 Fin du projet - 4h

5.1 Développement autonome.

A l'heure actuelle, votre projet doit être capable d'afficher un dongeon graphique. Si ce n'est pas le cas, vous pouvez vous resynchroniser avec l'avancé du TP en utilisant la solution en ligne à l'adresse suivante : `https://github.com/antoineTauvelENSEA/FISE_2024_2025_Dungeon_Crawler`

Au début de la séance, vous devez choisir un objectif spécifique en binôme, **et le faire valider par l'enseignant**. Vous avez quatre heures pour coder ce ou ces objectifs, il faudra donc prévoir du travail personnel supplémentaire. La notation de bonus des TD se fera essentiellement sur ces séances avec les critères suivants :

- L'implication dans les séances (50 % de la note finale). Seront notamment pris en compte l'assiduité, la ponctualité, l'implication, la durée des pauses etc.
- La qualité du code rendu (usage de polymorphisme, commentaire au format javadoc, création de classes pertinentes, diagramme de classe. Cela comptera pour 25 % de la note.
- Le nombre et la qualité des fonctionnalités prévu, pour 25 %.

Voici une liste non exhaustive de fonctionnalité possible. N'hésitez pas à proposer votre fonctionnalité. A total, il faut au moins viser 3 étoiles de difficultés.

Nom	Fonctionnalité à prévoir	Difficulté
Ecran titre	On rentre dans le jeu en cliquant un bouton	*
Game Over	A la mort du personnage, un écran indique le game over	*
Affichage du framerate	Un affichage sur l'écran affiche le framerate	*
Indicateur de vie	Une barre de vie diminue à chaque collision avec un énemie	*
Présence de piège statiques	Les pièges diminue la vie du héros	*
Invincibilité temporaire	A chaque collision, on reste invincible pendant 2 secondes	**
Timer	Le temps pour sortir du labyrinthe est chronométrée	**
Niveaux	Des portes permettent de passer d'un niveau à un autre	**
Capacité à courir	En modifiant les éléments du moteur graphique et physique, le héros peut courir sous certaines conditions (appui long, ou appui de CTRL par exemple)	**
Caméra	Une caméra décale les affichages de manière à suivre le héros dans un donjon plus vaste qu'un écran.	**
Ennemie à pattern simple	Des ennemies ont des pattern (ronde de surveillance ou vise le héros)	***
Ennemie à pattern complexe	Des ennemies ont des patterns complexes impliquant la visibilité en ligne direct sur le héros (ronde de surveillance ou charge vers le héros)	****
Attaque de contact	Le héros utilise une arme ou un sort de contact dans une direction	***
Attaque à distance	Le héros utilise un arc ou un sort dans une direction	****
Attaque de zone	Le héros pose une bombe qui explosera au bout de 3 secondes	****

Lors de la dernière séance, la dernière heure sera consacrée à une présentation de votre travail : un extrait de code pertinent et le résultat de votre travail.

At this point, your project should be capable of displaying a graphical dungeon. If this is not the case, you can resynchronize with the progress of the lab by using the online solution available at: `https://github.com/antoineTauvelENSEA/FISE_2024_2025_Dungeon_Crawler`

At the beginning of the session, you must choose a specific objective in pairs, **and have it validated by the instructor**. *You have four hours to code this objective, so you should plan for additional personal work. The bonus grading for the tutorials will primarily take place during these sessions with the following criteria:*

- *Engagement during the sessions (50% of the final grade). This will particularly take into account attendance, punctuality, involvement, duration of breaks, etc.*
- *Quality of the submitted code (use of polymorphism, comments in Javadoc format, creation of relevant classes, class diagrams). This will count for 25% of the grade.*
- *Number and quality of the planned features, for 25%.*

Here is a non-exhaustive list of possible features. Feel free to propose your own feature.
Overall, you should aim for at least 3 stars in difficulty.

Name	Feature to Implement	Difficulty
Title Screen	Enter the game by clicking a button	*
Game Over	When the character dies, a screen indicates the game over	*
Framerate Display	A display on the screen shows the framerate	*
Health Indicator	A health bar decreases with each collision with an enemy	*
Presence of Static Traps	Traps decrease the hero's health	*
Temporary Invincibility	After each collision, the hero remains invincible for 2 seconds	**
Timer	The time to escape the maze is timed	**
Levels	Doors allow passage from one level to another	**
Ability to Run	By modifying elements of the graphical and physical engines, the hero can run under certain conditions (long press or CTRL key press, for example)	**
Camera	A camera offsets the displays to follow the hero in a dungeon larger than one screen.	**
Simple Pattern Enemies	Enemies have patterns (patrolling or targeting the hero)	***
Complex Pattern Enemies	Enemies have complex patterns involving direct line visibility on the hero (patrolling or charging towards the hero)	****
Contact Attack	The hero uses a weapon or spell in a direction	***
Ranged Attack	The hero uses a bow or spell in a direction	****
Area Attack	The hero places a bomb that will explode after 3 seconds	****

During the last session, the final hour will be dedicated to a presentation of your work: a relevant code excerpt and the results of your work.