Projet d'informatique 2022-2023

Séance spéciale "bonnes pratiques"

03/03/23

UMONS
Université de Mons

Outline

UMONS
Université de Mons

Outline

UMONS
Université de Mons

Java vs Python

- En Python, on est obligé d'indenter le code et on n'a pas besoins d'accolades.
- En Java, l'indentation n'est pas obligatoire mais les accolades sont très importantes.
- S'il y a des accolades, pourquoi indenter ?
- Où doit-on placer les accolades ?

UMONS
Université de Mons

Exemple

```
public class Facto{public static facto(int n){
if(n==0){return 1;}else{return n*facto(n-1);}}}
```

UMONS
Université de Mons

Exemple

Ne surtout pas faire ça !

```
public class Facto{public static facto(int n){
if(n==0){return 1;}else{return n*facto(n-1);}}}
```

UMONS
Université de Mons

Exemple

- Il faut retourner à la ligne quand nécessaire !
- Et il faut aussi **indenter** !

C'est déjà mieux

```
public class Facto{
    public static facto(int n){
        if(n==0){
            return 1;}
        else{
            return n*facto(n-1);}}}
```

UMONS
Université de Mons

Exemple

- Ensuite, on applique un style de code :

Accolades ouvrantes sur la même ligne

```
public class Facto {
    public static facto(int n) {
        if(n==0) {
            return 1;
        }
        else {
            return n*facto(n-1);
        }
    }
}
```

Exemple

- Ensuite, on applique un style de code :

Accolades ouvrantes sur la ligne suivante

```
public class Facto
{
    public static facto(int n)
    {
        if(n==0)
        {
            return 1;
        }
        else
        {
            return n*facto(n-1);
        }
    }
}
```

Style de code

- Dans tous les cas, il faut indenter son code.
- En général, on décide d'un style de code et on garde le même dans tout le projet.
- Entre autres choses, un style de code définit :
  - la longueur des indentations (en nombre d'espaces),
  - l'endroit où on place les accolades ouvrantes,
  - le nombre de caractères sur une ligne,
  - ...

UMONS
Université de Mons

Comment choisir ?

- Il y a énormément de styles différents avec plus ou moins de ressemblances.
- Vous pouvez vous inspirer du style de codes sur Internet ou dans vos cours.
- Ou mélanger des styles pour créer le votre.
- Tant que votre code est bien indenté et facile à lire, le style que vous choisissez ne dépend que de vos préférences.
- Ne changez pas de style au milieu du code.

UMONS
Université de Mons

Outline

UMONS
Université de Mons

Pourquoi documenter ?

- On travaille sur un projet (*e.g.*, projet de master).
- Il faut l'arrêter pendant quelques mois (*e.g.*, à cause d'examens).
- Et puis il faut s'y remettre.
- Mais. . .

UMONS
Université de Mons

```
public final List<MouseBoxPosition> getBFSShortestPath() {
    List<MouseBoxPosition> path = new LinkedList<>();
    MouseBoxPosition start = mousePosition;
    if (isMouseWinnerPosition(start)) {
        return path;
    }
    MouseBoxPosition[][] aMarkedFromEdge = new
    ↪  MouseBoxPosition[MouseBoxPosition.NBR_COLUMNS][MouseBoxPosition.NBR_ROWS];
    int[][] aDistFromStart = new int[MouseBoxPosition.NBR_COLUMNS][MouseBoxPosition.NBR_ROWS];
    for (int[] aCol : aDistFromStart) {
        Arrays.fill(aCol, Integer.MAX_VALUE);
    }
    aDistFromStart[start.col][start.row] = 0;
    Queue<MouseBoxPosition> positionsToVisit = new LinkedList<>();
    positionsToVisit.add(start);
    MouseBoxPosition current;
    while (positionsToVisit.size() > 0) {
        current = positionsToVisit.remove();
    if (isMouseWinnerPosition(current)) {
        MouseBoxPosition prec = aMarkedFromEdge[current.col][current.row];
        while (prec != null) {
            ((LinkedList<MouseBoxPosition>) path).addFirst(current);
            current = prec;
            prec = aMarkedFromEdge[current.col][current.row];
        }
        return path;
    }
    Set<MouseBoxPosition> nextPositions = getLegitMousePositionsFromPosition(current);
    for (MouseBoxPosition next : nextPositions) {
        if (aDistFromStart[next.col][next.row] == Integer.MAX_VALUE) {
            aDistFromStart[next.col][next.row] = aDistFromStart[current.col][current.row] + 1;
            aMarkedFromEdge[next.col][next.row] = current;
            positionsToVisit.add(next);
        }
    }
}
return null;
```

UMONS
Université de Mons

Documenter

Pourquoi ?

- Facilite la compréhension du code pour une personne extérieure (par ex.: vos professeurs)
- Permet de se replonger plus facilement et rapidement dans du code après un certain laps de temps
- Aide à se rappeler à quoi sert une méthode, pourquoi avoir divisé ce paramètre par 2,...

Comment ?

- commentaires internes
- javadoc

UMONS
Université de Mons

Documenter

Pourquoi ?

- Facilite la compréhension du code pour une personne extérieure (par ex.: vos professeurs)
- Permet de se replonger plus facilement et rapidement dans du code après un certain laps de temps
- Aide à se rappeler à quoi sert une méthode, pourquoi avoir divisé ce paramètre par 2,...

Comment ?

- commentaires internes
- javadoc

UMONS
Université de Mons

Documenter

Pourquoi ?

- Facilite la compréhension du code pour une personne extérieure (par ex.: vos professeurs)
- Permet de se replonger plus facilement et rapidement dans du code après un certain laps de temps
- Aide à se rappeler à quoi sert une méthode, pourquoi avoir divisé ce paramètre par 2,...

Comment ?

- commentaires internes
- javadoc

UMONS
Université de Mons

Documenter

Pourquoi ?

- Facilite la compréhension du code pour une personne extérieure (par ex.: vos professeurs)
- Permet de se replonger plus facilement et rapidement dans du code après un certain laps de temps
- Aide à se rappeler à quoi sert une méthode, pourquoi avoir divisé ce paramètre par 2,...

Comment ?

- commentaires internes
- javadoc

UMONS
Université de Mons

Commentaires internes

- Utilisés pour documenter certaines parties de codes
- À utiliser pour donner certaines explications: par exemple pour savoir où on en est dans une série d'opérations au sein d'une méthode
- sur une ligne (//...) ou sur plusieurs lignes (/*...*/)

```
public static void main(String[] args){
    //Un premier commentaire
    //Un second commentaire

    /* Un bloc
    de
    commentaires
    */
}
```

UMONS
Université de Mons

Commentaires internes

- On les utilise pour documenter ce qui est utile au développeur, mais ne sert à rien à l'utilisateur.

```
public MouseMove(MouseBoxPosition from, MouseBoxPosition to) {
    // le déplacement est en croix si et seulement si
    // le carré de la longueur du déplacement est égal à 1
    if ((to.col - from.col) * (to.col - from.col) + (to.row -
    ↪  from.row) * (to.row - from.row) != 1) {
        throw new IllegalMoveException("Mouse must move to
        ↪  adjacent box (" + from + "," + to + ")");
    }
    this.from = from;
    this.to = to;
}
```

UMONS
Université de Mons

La javadoc

C'est quoi ?

1. Une syntaxe
2. Un outil qui produit une documentation navigable
3. Par abus de langage la documentation elle-même

Que contient la documentation ?

- une liste de package/classes/interfaces
- pour chaque classe ou interface les méthodes et attributs publics[1]
- des descriptions textuelles
- des informations relatives aux attributs, méthodes, classes (indiqués par des "tags")

[1] Typiquement ce qu'on retrouve dans l'API java
`http://docs.oracle.com/javase/7/docs/api/`

UMONS Université de Mons

La javadoc

C'est quoi ?

1. Une syntaxe
2. Un outil qui produit une documentation navigable
3. Par abus de langage la documentation elle-même

Que contient la documentation ?

- une liste de package/classes/interfaces
- pour chaque classe ou interface les méthodes et attributs publics[1]
- des descriptions textuelles
- des informations relatives aux attributs, méthodes, classes (indiqués par des "tags")

[1] Typiquement ce qu'on retrouve dans l'API java
http://docs.oracle.com/javase/7/docs/api/

UMONS Université de Mons

La javadoc

C'est quoi ?

1. Une syntaxe
2. Un outil qui produit une documentation navigable
3. Par abus de langage la documentation elle-même

Que contient la documentation ?

- une liste de package/classes/interfaces
- pour chaque classe ou interface les méthodes et attributs publics[1]
- des descriptions textuelles
- des informations relatives aux attributs, méthodes, classes (indiqués par des "tags")

[1] Typiquement ce qu'on retrouve dans l'API java `http://docs.oracle.com/javase/7/docs/api/`

UMONS Université de Mons

Les tags javadoc

Ils sont au nombre de neuf[2] et permettent de fournir des informations supplémentaires:

1. @param : les paramètres d'une méthode
2. @return : l'objet retourné par une méthode
3. @author : l'auteur de la classe
4. @throws : les exceptions propagées
5. ...

UMONS
Université de Mons

[2] Les 4 présentés sont les plus importants

Écrire la javadoc

- La liste des méthodes, classes et attributs est générée automatiquement
- Il faut donc juste rajouter les informations complémentaires
- Ces informations sont écrites immédiatement avant le nom de la classe/méthode/variable
- Les balises "/**" et "*/" sont utilisées pour encadrer les blocs javadocs
- Chaque ligne du bloc peut être préfixée par un "*"

UMONS
Université de Mons

Écrire la javadoc

```java
/**
* Initialise un déplacement de souris à partir de sa position
* initiale et de sa position finale.
*
* @param from La position initiale de la souris.
* @param to La position finale de la souris.
* @throws IllegalMoveException si le déplacement est impossible
*         (sans tenir compte de la présence de murs).
*/
public MouseMove(MouseBoxPosition from, MouseBoxPosition to) {
```

UMONS
Université de Mons

Écrire la javadoc

```
/**
* Renvoie un chemin le plus court en nombre de déplacements de souris
* pour atteindre un fromage. Ce chemin ne contient pas la position
* de départ, mais contient la position d'arrivée.
* <p>
* L'algorithme utilisé est celui de parcours de graphe en largeur
* (ou BFS, pour Breadth First Search en anglais).
* <p>
* Ref: https://en.wikipedia.org/wiki/Breadth-first_search
*
* @return un chemin le plus court en nombre de déplacements de souris
* pour atteindre un fromage ou null s'il n'y a pas de chemin possible
* (le chemin ne contient pas la position de départ, mais contient
* la position d'arrivée).
*/
// final pour appel depuis constructeur (warning)
public final List<MouseBoxPosition> getBFSShortestPath() {
```

UMONS
Université de Mons

Format habituel de la javadoc

- La première phrase est un court résumé de la méthode
- On explique ensuite la méthode et on donne des détails éventuels
- On laisse une ligne vide avant d'ajouter les tags.
- Pour chaque tag, on donne l'argument si nécessaire (*e.g.,* le nom du paramètre).
- La description associée au tag peut s'étendre sur plusieurs lignes jusqu'au prochain tag ou à la fin du commentaire.

UMONS
Université de Mons

Comment bien documenter ?

- Expliquer à quoi ça sert et comment l'utiliser.
- Éventuellement d'autres informations utiles comme la complexité d'une méthode.
- Utiliser les tags de la javadoc pour plus de lisibilité.
- Dans la javadoc, pas de détails techniques sans intérêt pour l'utilisateur.
- Dans le code, expliquer les parties techniques, d'où viennent les formules... *i.e.,* ce qui n'est pas facile à comprendre.

UMONS
Université de Mons

Générer la javadoc

1. Dans IntelliJ IDEA : Tools > Generate JavaDoc;
2. dans Eclipse : Projet > Générer la Javadoc;
3. en console :

```
javadoc -d <destination> -sourcepath <dossierSource> <fichiers/packages>
```

UMONS
Université de Mons

Petite parenthèse

```
/**
* Renvoie un chemin le plus court en nombre de déplacements de souris
* pour atteindre un fromage. Ce chemin ne contient pas la position
* de départ, mais contient la position d'arrivée.
* <p>
* L'algorithme utilisé est celui de parcours de graphe en largeur
* (ou BFS, pour Breadth First Search en anglais).
* <p>
* Ref: https://en.wikipedia.org/wiki/Breadth-first_search
*
* @return un chemin le plus court en nombre de déplacements de souris
* pour atteindre un fromage ou null s'il n'y a pas de chemin possible
* (le chemin ne contient pas la position de départ, mais contient
* la position d'arrivée).
*/
// final pour appel depuis constructeur (warning)
public final List<MouseBoxPosition> getBFSShortestPath() {
```

- Qu'est-ce que cet algorithme ?

UMONS Université de Mons

L'algorithme de parcours en largeur

- Il s'agit d'un algorithme de théorie des graphes.
- On explore tous les plus courts chemins potentiels.
- À chaque étape, on élargit la zone déjà explorée.
- Par opposition, le parcours en profondeur explore d'abord un chemin en entier avant d'explorer les autres.

UMONS
Université de Mons

L'algorithme de parcours en largeur

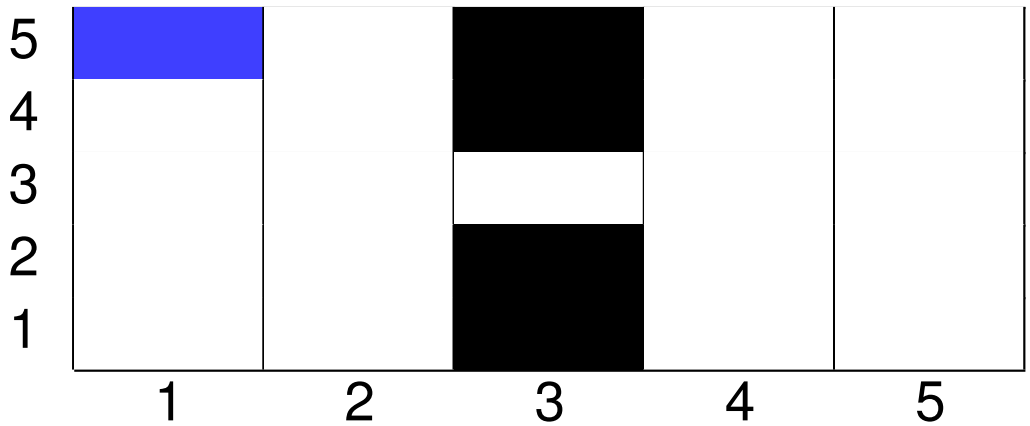
- L'idée est de décomposer la carte en trois parties :
  - la partie déjà visitée,
  - la frontière
  - et la partie à visiter.
- À chaque itération, on veut ajouter des cases de la frontière à la partie explorée et on ajoute des cases non explorées à la frontière.
- Quand on a plus de cases à explorer, on a fini.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(1, 5)$
Prédécesseurs :

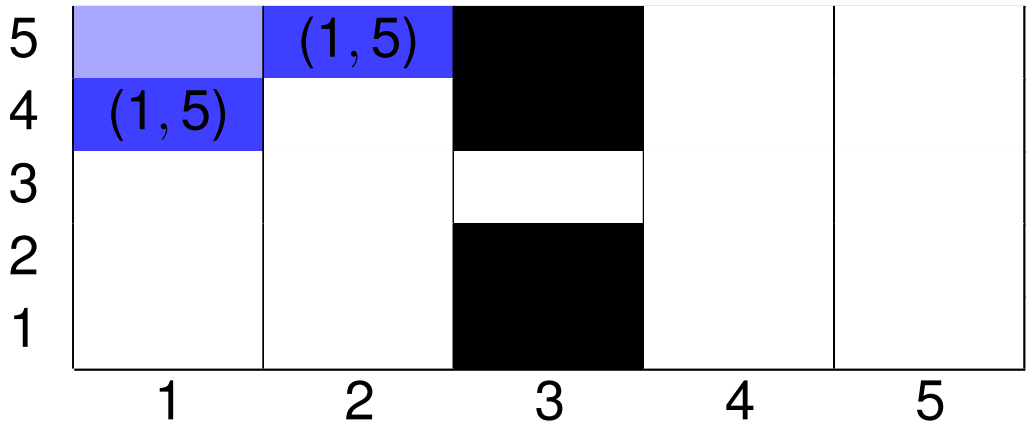


UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(2, 5), (1, 4)$
Prédécesseurs :

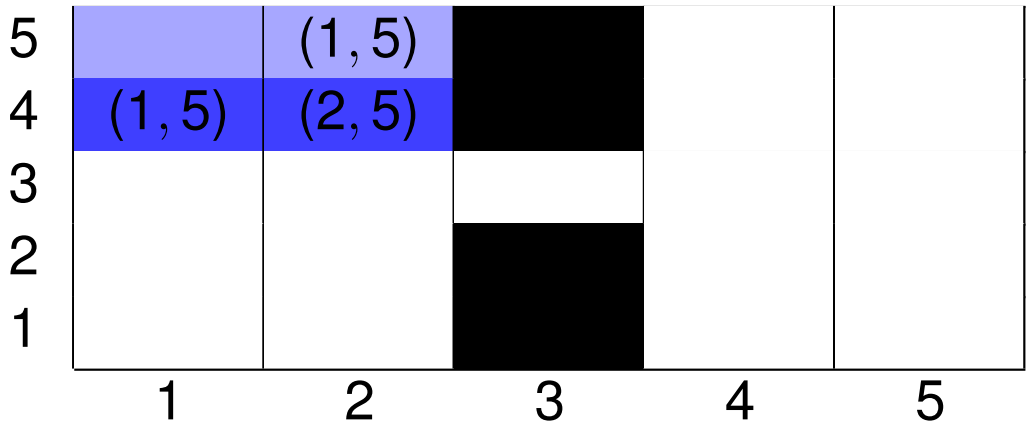


UMONS
Université de Mons

Illustration

Comment aller de $(1,5)$ à $(5,1)$?

Frontière : $(1,4),(2,4)$
Prédécesseurs :

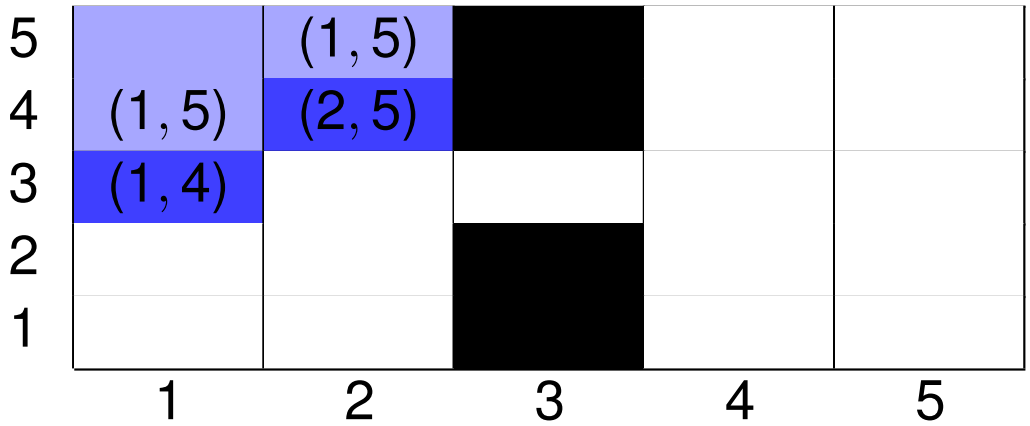


UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(2, 4), (1, 3)$
Prédécesseurs :



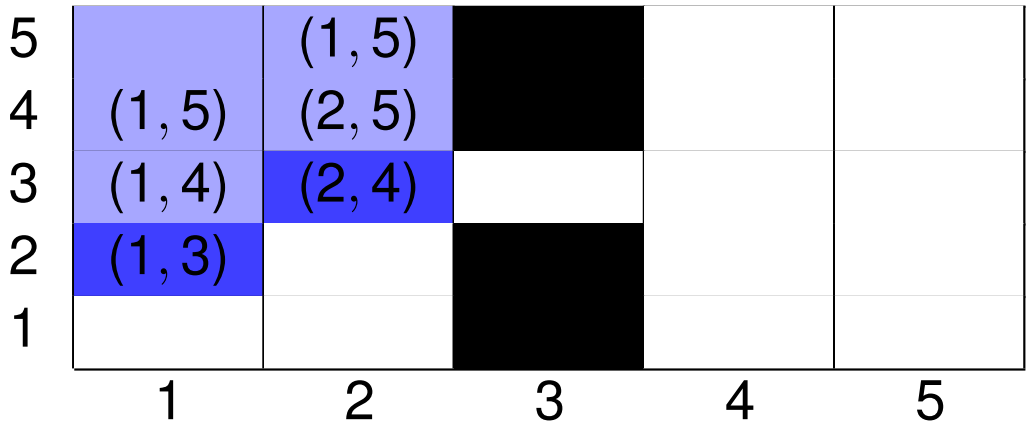
Allons un peu plus vite.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(2, 3), (1, 2)$
Prédécesseurs :



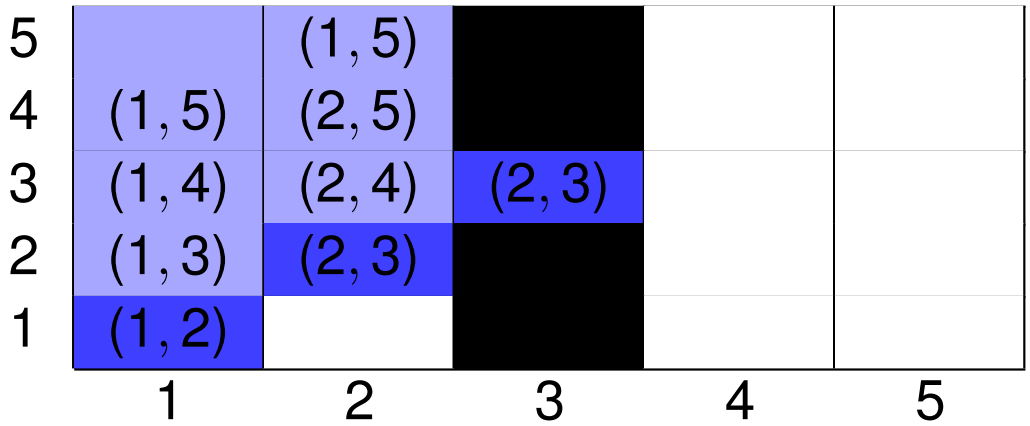
Allons un peu plus vite.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(3, 3), (2, 2), (1, 1)$
Prédécesseurs :



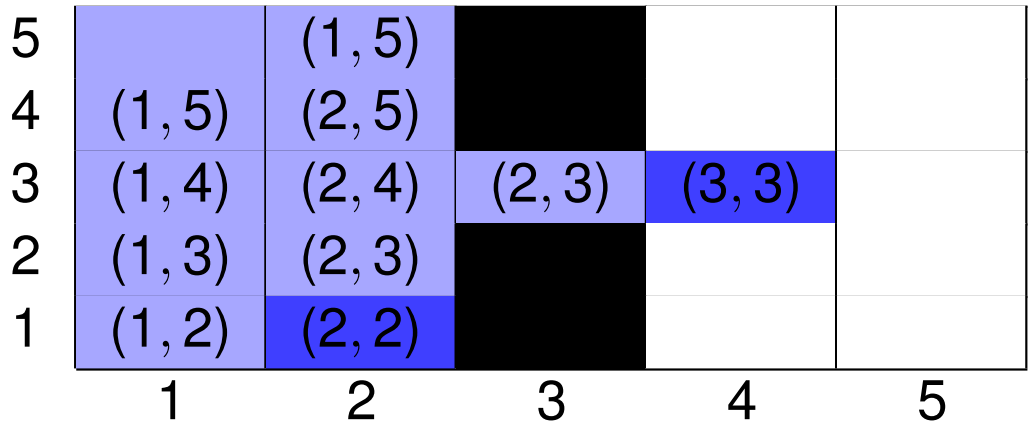
Allons un peu plus vite.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(4, 3), (2, 1)$
Prédécesseurs :



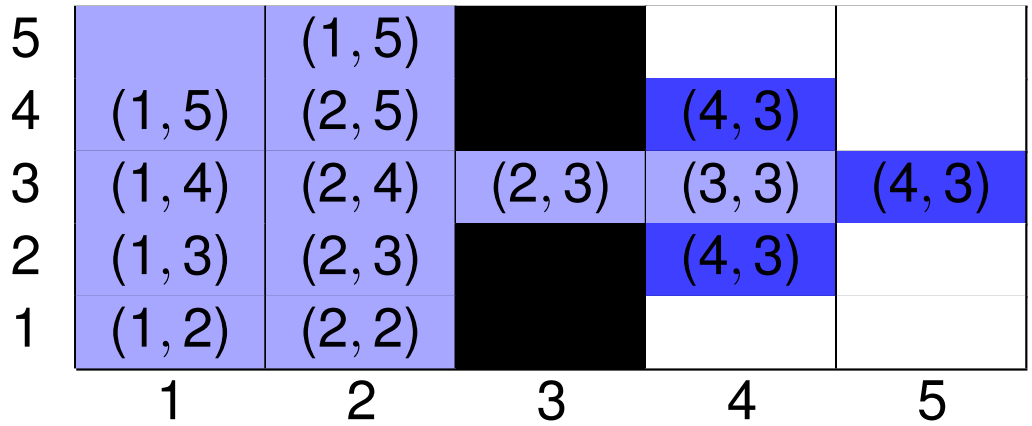
Allons un peu plus vite.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(4, 2), (5, 3), (4, 4)$
Prédécesseurs :



Allons un peu plus vite.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(4, 1), (4, 2), (5, 4), (4, 5)$
Prédécesseurs :

5		$(1, 5)$		$(4, 4)$	
4	$(1, 5)$	$(2, 5)$		$(4, 3)$	$(5, 3)$
3	$(1, 4)$	$(2, 4)$	$(2, 3)$	$(3, 3)$	$(4, 3)$
2	$(1, 3)$	$(2, 3)$		$(4, 3)$	$(4, 2)$
1	$(1, 2)$	$(2, 2)$		$(4, 2)$	
	1	2	3	4	5

Allons un peu plus vite.

UMONS Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Frontière : $(4, 2), (5, 4), (4, 5)$
Prédécesseurs :

5		$(1, 5)$		$(4, 4)$	
4	$(1, 5)$	$(2, 5)$		$(4, 3)$	$(5, 3)$
3	$(1, 4)$	$(2, 4)$	$(2, 3)$	$(3, 3)$	$(4, 3)$
2	$(1, 3)$	$(2, 3)$		$(4, 3)$	$(4, 2)$
1	$(1, 2)$	$(2, 2)$		$(4, 2)$	$(4, 1)$
	1	2	3	4	5

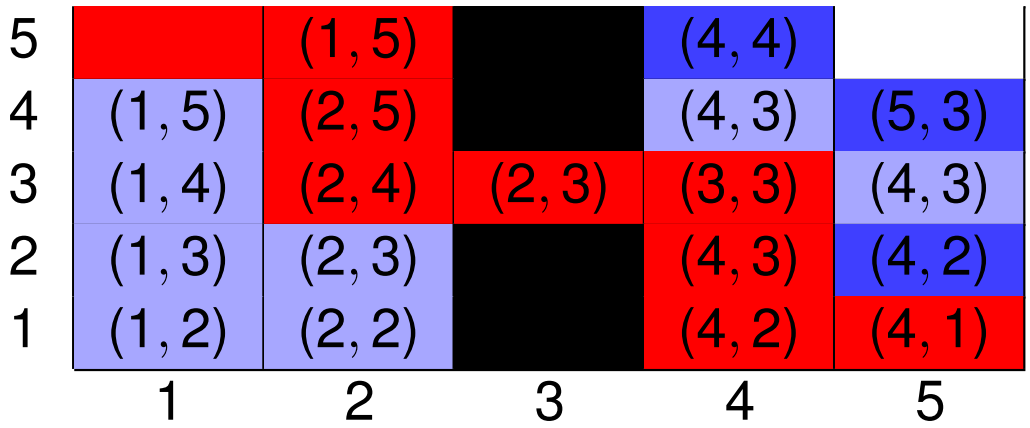
On a atteint notre objectif.

UMONS
Université de Mons

Illustration

Comment aller de $(1, 5)$ à $(5, 1)$?

Chemin de longueur 8 :
$(5, 5), (4, 1), (4, 2), (4, 3), (3, 3), (2, 3), (2, 4), (2, 5), (1, 5)$
Prédécesseurs :

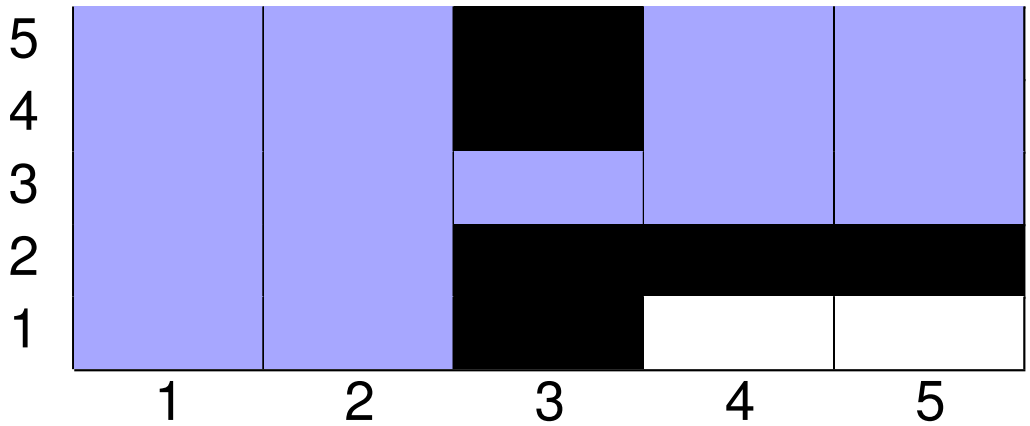


On a atteint notre objectif.

UMONS
Université de Mons

Illustration

Frontière :



- Pour juste vérifier l'atteingnabilité, pas besoin de prédécesseurs.
- Si la frontière est vide et qu'on a pas atteint l'objectif, il n'y a pas de chemin.

UMONS
Université de Mons

Outline

UMONS
Université de Mons

- Quand on programme en Java, on peut vite se retrouver avec beaucoup de classes :

```
Pair.java Utilities.java
AAction.java
RoleChooseAction.java
MouseMoveAction.java
AMoveAction.java
HunterMoveAction.java
ThrowDiceAction.java
RoleChooseEvent.java
DiceEvent.java
MouseMoveEvent.java
AMoveEvent.java
HunterMoveEvent.java
AEvent.java
InitializedEvent.java
IAsyncPlayerObserver.java
TrivialPlayerIA.java
RandomPlayerIA.java
HumanPlayer.java IPlayer.java
PlayerIAAsyncAdapter.java
APlayer.java IAsyncPlayer.java
IllegalMoveException.java
IllegalPositionException.java
IllegalActionException.java
EnumRole.java WallMove.java
AMove.java
MouseBoxPosition.java
Dice.java WallBoxPosition.java
Board.java MouseMove.java
EnumGameState.java
GameModel.java
```

- On préfère donc les grouper.

UMONS
Université de Mons

Les packages

C'est quoi ?

Un groupe de classes

Pourquoi ?

- Organiser les classes
- Grouper les classes en modules
- Gagner en lisibilité
- Permettre à deux classes d'avoir le même nom (si dans deux packages différents)
- Faciliter les imports (importer tout un package plutôt que chaque classe séparément)

UMONS
Université de Mons

Les packages

C'est quoi ?

Un groupe de classes

Pourquoi ?

- Organiser les classes
- Grouper les classes en modules
- Gagner en lisibilité
- Permettre à deux classes d'avoir le même nom (si dans deux packages différents)
- Faciliter les imports (importer tout un package plutôt que chaque classe séparément)

UMONS
Université de Mons

Les packages

C'est quoi ?

Un groupe de classes

Pourquoi ?

- Organiser les classes
- Grouper les classes en modules
- Gagner en lisibilité
- Permettre à deux classes d'avoir le même nom (si dans deux packages différents)
- Faciliter les imports (importer tout un package plutôt que chaque classe séparément)

UMONS
Université de Mons

Les packages

C'est quoi ?

Un groupe de classes

Pourquoi ?

- Organiser les classes
- Grouper les classes en modules
- Gagner en lisibilité
- Permettre à deux classes d'avoir le même nom (si dans deux packages différents)
- Faciliter les imports (importer tout un package plutôt que chaque classe séparément)

UMONS
Université de Mons

Quelques conseils pour les packages

- Ne pas en faire trop sinon on perd l'intérêt de grouper les classes.
- Ni trop peu pour la même raison.
- Chaque package représente une notion, un principe (*e.g.,* la gui, le moteur de jeu)
- Dans un package, seulement des classes qui ont un rapport entre elles.

UMONS
Université de Mons

Ajouter une classe dans un package

2 modifications

- ajouter la clause **package** suivie du nom du package avant le nom de la classe
- créer sur l'ordinateur un dossier qui correspond au nom du package et y placer les fichiers `.java`

```
package be.ac.umons.projet;
public class Test{
    ...
}
```

UMONS
Université de Mons

Les noms de package

- Nom unique
- On utilise souvent les noms de domaines inversés.
  Exemple : `be.ac.umons.projet`
- Organisation hiérarchique
  Sur le pc on aura donc les dossiers `be > ac > umons > projet`.

UMONS
Université de Mons

Exemple d'une bonne structure

- package général (*p. ex.,* `be.ac.umons.projet`) :
  - `console` : contient les classes utiles pour la partie console du projet
  - game :
    - `action` : classes représentant des actions (effectuées par les joueurs)
    - `event` : classes représentant des évènements
    - `exception` : exceptions spécifiques pour le projet
    - `model` : implémentation du modèle de jeu (plateau, pions, ...)
    - `player` : classes pour les joueurs (IAs et humains)
  - `gui` : contient les classes utiles pour l'interface graphique

UMONS
Université de Mons

Outline

UMONS
Université de Mons

Compiler un projet

- Avec quelques classes, compiler en console est facile.
- Quand on réalise un projet, les commandes peuvent devenir plus longues.

```
$ javac -encoding Latin1 A.java B.java C.java D.java
↪  E.java -cp Lib1:Lib2:.
$ java main -cp Lib1:Lib2:.
$ javadoc -d ../../doc ccm.console ccm.game.action
↪  ccm.game.event ccm.game.exception ccm.game.model
↪  ccm.game.player ccm.gui
```

- Il serait intéressant de ne pas devoir réécrire tout ça à chaque fois...

UMONS
Université de Mons

Gradle

Build tool

C'est quoi ?

- Outil pour la construction d'applications (compilation, exécution,...)[3]
- Propose à l'utilisateur un ensemble de tâches déjà prévues.
- Configuration dans un fichier **build.gradle**

Pourquoi ?

- Permettre à un utilisateur de compiler et exécuter un programme sans avoir à se soucier des *classpath*, dossiers sources ou destinations, options spécifiques aux compilateurs,...

UMONS
Université de Mons

[3] https://gradle.org/

Gradle

Build tool

C'est quoi ?

- Outil pour la construction d'applications (compilation, exécution,...)[3]
- Propose à l'utilisateur un ensemble de tâches déjà prévues.
- Configuration dans un fichier **build.gradle**

Pourquoi ?

- Permettre à un utilisateur de compiler et exécuter un programme sans avoir à se soucier des *classpath*, dossiers sources ou destinations, options spécifiques aux compilateurs,...

UMONS
Université de Mons

[3] `https://gradle.org/`

Gradle

Build tool

C'est quoi ?

- Outil pour la construction d'applications (compilation, exécution,...)[3]
- Propose à l'utilisateur un ensemble de tâches déjà prévues.
- Configuration dans un fichier **build.gradle**

Pourquoi ?

- Permettre à un utilisateur de compiler et exécuter un programme sans avoir à se soucier des *classpath*, dossiers sources ou destinations, options spécifiques aux compilateurs,...

UMONS
Université de Mons

[3] `https://gradle.org/`

Gradle

Build tool

C'est quoi ?

- Outil pour la construction d'applications (compilation, exécution,...)[3]
- Propose à l'utilisateur un ensemble de tâches déjà prévues.
- Configuration dans un fichier **build.gradle**

Pourquoi ?

- Permettre à un utilisateur de compiler et exécuter un programme sans avoir à se soucier des *classpath*, dossiers sources ou destinations, options spécifiques aux compilateurs,...

UMONS
Université de Mons

[3] `https://gradle.org/`

Idées de Gradle

- Compiler, exécuter, lancer les tests, . . . utilisent toujours les même commandes mais avec des paramètres différents.
- Plutôt que de demander ces paramètres à l'utilisateur, Gradle utilise des valeurs choisies par conventions.
- Il sait donc comment compiler, exécuter, . . . Il faut juste lui donner quelques informations.
- Avantages :
  - Tout le monde travaille de la même façon.
  - Pas besoin de s'occuper des dépendances à la main.
  - Peu de configuration nécessaire.

UMONS
Université de Mons

Structure d'un projet Gradle

- Comme Gradle utilise des conventions, il s'attend à ce que le projet respecte une structure particulière :

```
<votre projet> ..... dossier de votre projet
├── build.gradle
└── src
    ├── main
    │   ├── java ..... contient les fichiers .java
    │   └── resources ..... contient les resources (images, ...)
    └── test
        └── java ..... contient les fichiers .java pour les tests
```

UMONS
Université de Mons

Un build.gradle minimal

```
plugins {
    id 'java' // Utilise le plugin Java
    id 'application' // On développe une application
}
repositories {
    // Endroit où trouver les dépendances
    // (serveur jcenter.bintray.com)
    jcenter()
}
dependencies {
    //  Dépendance pour envoyer des mails
    implementation 'com.sun.mail:javax.mail:1.6.2'
}
application {
    // Classe principale à exécuter
    mainClassName = 'monprojet.ClassePrincipale'
}
```

UMONS
Université de Mons

Les dépendances

- Lorsqu'on a besoin d'utiliser des librairies Java, il suffit de le dire à Gradle et il va se charger de les télécharger.
- Pour chaque librairie, il faut donner trois informations :
  - le groupe (ex: mes-projets),
  - le nom (ex: projet-ba1),
  - et la version (ex: 2.6).
- On peut simplifier en <groupe>:<nom>:<version>.
- On précise aussi si la dépendance est pour l'implémentation du projet pour juste pour les tests avec les mots-clés `implementation` et `testImplementation`[4].

[4] Il existe aussi runtimeOnly et testRuntimeOnly pour les dépendances qui ne sont pas nécessaire pour compiler mais seulement pour l'exécution.

Utilisation de Gradle

- Une fois le projet bien configuré, s'il respecte la structure imposée par Gradle, les commandes suivantes sont disponibles :

gradle build	compile le programme.
gradle test	exécute les tests unitaires.
gradle run	exécute le programme.
gradle clean	supprime les fichiers .class
gradle tasks	liste toutes les commandes fournies par Gradle.

UMONS
Université de Mons

Outline

UMONS
Université de Mons

Où en est-on ?

- On a un texte bien documenté.
- Les classes sont bien structurées et groupées en packages.
- On a un outil pour nous aider à compiler, générer la doc, . . .
- Mais est-ce qu'il y a des bugs ?

UMONS
Université de Mons

Comment tester ?

- On pourrait juste tester par l'interface graphique.
  - Mais est-ce qu'on peut tout tester comme ça ?
  - Combien de scénarios différents faut-il tester ?
- On peut écrire un programme de test.
  - C'est déjà mieux, mais on aura beaucoup de résultats à lire.
  - Et surtout, que faut-il tester ?

UMONS
Université de Mons

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS
Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Un test unitaire

C'est quoi ?

- Un test employé pour évaluer de manière automatique le bon fonctionnement d'une "unité" de logiciel[5]

Pourquoi ?

- Vérifier qu'un bout de code (par exemple une méthode) effectue correctement sa tâche, après avoir été modifié, quand on change de paramètres, ...
- Évaluer l'impact de certaines modifications sur d'autres parties de code

Comment ?

- Employer le système **JUnit** (voir chapitre 7 du cours d'algo 2)

UMONS Université de Mons

[5] petite partie de code remplissant une tâche spécifique

Que tester?

- Éviter d'évaluer des choses très simples. Par exemple, les accesseurs.
- Souvent utile de tester les valeurs limites

Mise en pratique

Supposez que vous développez un petit jeu de courses de voitures via interface graphique. Qu'allez-vous vérifier pour être sûr qu'une voiture avance correctement?

- Le test unitaire ne doit pas lancer l'interface graphique
- Donnez une position initiale, une vitesse, un laps de temps et une direction.
- Calculez sur papier la position finale.
- Demandez au test unitaire de comparer ce résultat à celui obtenu par la méthode

Exemple

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions; //on importe la classe contenant les
                                         // méthodes assert*

public class MyTests
{
    @Test //annotation pour indiquer qu'il s'agit d'une méthode de test
    public void driveTest() {
        Car mustang = new Car();
        Position finalPos = new Position(19,20); // la position calculée sur "papier"
        Position startPos = new Position(1,2);
        int direction = 30;
        int speed = 50;
        int time = 20;
        //1er test
        Position res = mustang.drive(startPos,direction,speed,time)
        Assertions.assertTrue(finalPos.getX()==res.getX() && finalPos.getY()==res.getY(),"Tout
        ↪  roule");
    }

    @Test
    public void secondTest(){
        ...
    }
}
```

JUnit

- Configurer Gradle pour utiliser JUnit5[6]
- Import de org.junit.jupiter.api.Test dans votre classe de test
- Annotation des méthodes de tests avec le mot clé **@Test**
- Emploi de méthodes spécifiques de type assert: assertTrue, assertFalse, assertEquals,...[7]
- Compilation et exécution :
  gradle test

[6] voir exemples ou `https://junit.org/junit5/docs/current/user-guide/#running-tests-build-gradle`.

[7] org.junit.jupiter.api.Assertions

UMONS Université de Mons

Outline

UMONS
Université de Mons

Tester que ça fonctionne

- Préparer un dossier avec uniquement le code source, le rapport au format pdf, le fichier build.gradle et les fichiers nécessaires au fonctionnement (images, niveaux, . . .).
- Attention à bien conserver la bonne structure pour Gradle.
- Sur une machine de la salle Escher, exécuter :
  - gradle build
  - gradle test
  - gradle run
  - gradle clean
- Après chaque commande, vérifier que tout a fonctionné (pas de fichiers .class après un clean, pas d'erreur de compilation, . . .)

UMONS
Université de Mons

Avant d'archiver

- Dans le dossier utilisé, vérifier qu'il ne manque rien.
- Vérifier qu'il n'y a plus de fichiers .class.
- S'assurer que le nom du fichier est correct.
- Et créer une archive.

UMONS
Université de Mons

Choix du format

Pourquoi zip ?

- Disponible nativement sous Windows, Mac et Linux.
- Format libre (on ne dépend pas des décisions des propriétaires)

Pourquoi pas rar ?

- Pas disponible partout par défaut (pas sous Linux).
- Format propriétaire (risque que le propriétaire décide de ne plus distribuer son programme)

Le format tar.gz est aussi libre, mais moins répandu.

UMONS
Université de Mons

Comment archiver ?

Windows

- clic droit sur le dossier puis choisir
  `Envoyer vers → Dossier compressé`

Mac OS et Ubuntu

- clic droit sur le dossier puis choisir
  `Compresser`

Bonus : en console

- `zip -r <votre archive>.zip <votre dossier>`

UMONS
Université de Mons

Quelques conseils pour finir

- Évitez de tester tous les cas dans le même test unitaire. C'est plus difficile à débugger.
- Si vos méthodes deviennent longues, vous pouvez les découper en plus petites méthodes pour rendre le code plus lisible.
- Pas de copier-coller ! S'il y a une erreur, vous devez la corriger plusieurs fois.
- Si vous voulez faire un copier-coller, faites plutôt une fonction.
- Écrivez les commentaires en même temps que le code. Vous aurez moins de travail après.
- Et surtout, testez votre code sur les machines des salles informatiques avant de le rendre (chemins absolus, encodage, ...).

UMONS
Université de Mons