Programmation Orientée Objet

2 – Mise en œuvre

P. Berthomé

INSA Centre Val de Loire
Département STI — 3 ème année

26 octobre 2017

Objectifs du cours

Objectifs

- Comprendre l'organisation d'un programme Java

Objectifs du cours

Objectifs

- Comprendre l'organisation d'un programme Java
- Quelques *bonnes pratiques*

Objectifs du cours

Objectifs

- Comprendre l'organisation d'un programme Java
- Quelques *bonnes pratiques*
- Mise en place des schémas UML :

Objectifs du cours

Objectifs

- Comprendre l'organisation d'un programme Java
- Quelques *bonnes pratiques*
- Mise en place des schémas UML :
  - ce qui passe

Objectifs du cours

Objectifs

- Comprendre l'organisation d'un programme Java
- Quelques *bonnes pratiques*
- Mise en place des schémas UML :
  - ce qui passe
  - le reste

Premier Programme

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

Premier Programme

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

Quelques remarques

- On écrit directement une classe

Premier Programme

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

Quelques remarques

- On écrit directement une classe
- Méthode `main`

Premier Programme

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

Quelques remarques

- On écrit directement une classe
- Méthode `main`
- Nom du fichier `Hello.java`

Compilation et exécution

Compilation

- dans le répertoire spécifique

Compilation et exécution

Compilation

- dans le répertoire spécifique
- `javac Hello.java`

Compilation et exécution

Compilation

- dans le répertoire spécifique
- `javac Hello.java`
- Produit le fichier `Hello.class`

Compilation et exécution

Compilation

- dans le répertoire spécifique
- `javac Hello.java`
- Produit le fichier `Hello.class`

Compilation et exécution

Compilation

- dans le répertoire spécifique
- `javac Hello.java`
- Produit le fichier `Hello.class`

Exécution

- `java Hello`

Compilation et exécution

Compilation

- dans le répertoire spécifique
- `javac Hello.java`
- Produit le fichier `Hello.class`

Exécution

- `java Hello`

Environnements intégrés

- `Eclipse`
- `Netbeans`
- ...

Cela ressemble un peu au C

Ressemblances

- Types élémentaires : `int`, `char`, `float`, ...

Cela ressemble un peu au C

Ressemblances

- Types élémentaires : `int`, `char`, `float`, ...
- Les structures de contrôle : `for`, `if`, `while`, ...

Cela ressemble un peu au C

Ressemblances

- Types élémentaires : `int`, `char`, `float`, ...
- Les structures de contrôle : `for`, `if`, `while`, ...

Cela ressemble un peu au C

Ressemblances

- Types élémentaires : `int`, `char`, `float`, ...
- Les structures de contrôle : `for`, `if`, `while`, ...

Cependant

- On peut déclarer les variables quand on en a besoin

Cela ressemble un peu au C

Ressemblances

- Types élémentaires : `int`, `char`, `float`, ...
- Les structures de contrôle : `for`, `if`, `while`, ...

Cependant

- On peut déclarer les variables quand on en a besoin
- Attention ! Portée des variables

Cela ressemble un peu au C

Ressemblances

- Types élémentaires : `int`, `char`, `float`, ...
- Les structures de contrôle : `for`, `if`, `while`, ...

Cependant

- On peut déclarer les variables quand on en a besoin
- Attention ! Portée des variables
- Attention ! Toutes les variables doivent être initialisées avant utilisation.

Exemple

```
int j;
int newj = j;
```

Exemple

```
int j;
int newj = j;
```

Erreur du compilateur

```
The local variable j may not have been initialized
```

Exemple

```
int j;
int newj = j;
```

Erreur du compilateur

The local variable j may not have been initialized

```
int j;
if (a%2==0) j=3; else j=5;
int newj = j;
```

Ce n'est pas du Java

Valide simplement en C

- Les pointeurs : `char *`
- Les adresses : `&var`

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`
- Un certain nombre de classes contenues dans l'API

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`
- Un certain nombre de classes contenues dans l'API
  - `String`, `StringBuffer`, ...

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`
- Un certain nombre de classes contenues dans l'API
  - `String`, `StringBuffer`, ...
  - Plusieurs milliers de classes spécifiques

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`
- Un certain nombre de classes contenues dans l'API
  - `String`, `StringBuffer`, ...
  - Plusieurs milliers de classes spécifiques

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`
- Un certain nombre de classes contenues dans l'API
  - `String`, `StringBuffer`, ...
  - Plusieurs milliers de classes spécifiques

Passage de paramètre

- Pour les types simples, passage par valeur (IN)

Nouveautés Java

Types

- Un type booléen `boolean = true | false | expression booléenne`
- Un certain nombre de classes contenues dans l'API
  - `String`, `StringBuffer`, ...
  - Plusieurs milliers de classes spécifiques

Passage de paramètre

- Pour les types simples, passage par valeur (IN)
- Pour les objets passage par référence (IN/OUT)

Arguments du main

Paramètres

- `public static void main(String[] args) :`

Arguments du main

Paramètres

- `public static void main(String[] args)` :
- `args` : tableau de `String`

Arguments du main

Paramètres

- `public static void main(String[] args)` :
- `args` : tableau de `String`
- Le nombre d'arguments : `args.length`

Arguments du main

Paramètres

- `public static void main(String[] args)` :
- `args` : tableau de `String`
- Le nombre d'arguments : `args.length`
- `args[0]` : le premier argument

Arguments du main

Paramètres

- `public static void main(String[] args)` :
- `args` : tableau de `String`
- Le nombre d'arguments : `args.length`
- `args[0]` : le premier argument

Arguments du main

Paramètres

- `public static void main(String[] args)` :
- `args` : tableau de `String`
- Le nombre d'arguments : `args.length`
- `args[0]` : le premier argument

Utilisation

- En ligne de commande :

```
java monProg arg0 arg1 arg2
```

Arguments du main

Paramètres

- `public static void main(String[] args)` :
- `args` : tableau de `String`
- Le nombre d'arguments : `args.length`
- `args[0]` : le premier argument

Utilisation

- En ligne de commande :
  `java monProg arg0 arg1 arg2`
- Avec `Eclipse`, onglet spécifique

Sortie

`System.out`

- Sortie standard

Sortie

`System.out`

- Sortie standard
- Méthodes principales :

Sortie

`System.out`

- Sortie standard
- Méthodes principales :
  - `print`, `println`

Sortie

`System.out`

- Sortie standard
- Méthodes principales :
  - `print`, `println`
  - `write`

Sortie

`System.out`

- Sortie standard
- Méthodes principales :
  - `print`, `println`
  - `write`

Sortie

`System.out`

- Sortie standard
- Méthodes principales :
  - `print`, `println`
  - `write`

Manipulation des `String`

- Ne peuvent pas être modifiés en tant qu'objets

Sortie

`System.out`

- Sortie standard
- Méthodes principales :
  - `print`, `println`
  - `write`

Manipulation des `String`

- Ne peuvent pas être modifiés en tant qu'objets
- Concaténation : opérateur `+`

Entrées

Principe

- Objets spécifiques gérant la notion de flots

Entrées

Principe

- Objets spécifiques gérant la notion de flots
- Différentes possibilités (BufferedReader, Scanner, ...)

Entrées

Principe

- Objets spécifiques gérant la notion de flots
- Différentes possibilités (BufferedReader, Scanner, ...)
- Gestion des problèmes par des exceptions.

Entrées

Principe

- Objets spécifiques gérant la notion de flots
- Différentes possibilités (BufferedReader, Scanner, ...)
- Gestion des problèmes par des exceptions.

Entrées

Principe

- Objets spécifiques gérant la notion de flots
- Différentes possibilités (BufferedReader, Scanner, ...)
- Gestion des problèmes par des exceptions.

```
Scanner sc = new Scanner(System.in);
Integer i= null;
i = sc.nextInt();
System.out.println("Le nombre lu est : " + i);
sc.close();
```

Principes généraux

Classes et Objets

- Définition d'un type particulier

Principes généraux

Classes et Objets

- Définition d'un type particulier
- UML :

Principes généraux

Classes et Objets

- Définition d'un type particulier
- UML :
  - Conception des classes

Principes généraux

Classes et Objets

- Définition d'un type particulier
- UML :
  - Conception des classes
  - Spécification des méthodes

Principes généraux

Classes et Objets

- Définition d'un type particulier
- UML :
  - Conception des classes
  - Spécification des méthodes
- Java :

Principes généraux

Classes et Objets

- Définition d'un type particulier
- UML :
  - Conception des classes
  - Spécification des méthodes
- Java :
  - Représentation *physique* des données

Principes généraux

Classes et Objets

- Définition d'un type particulier
- UML :
  - Conception des classes
  - Spécification des méthodes
- Java :
  - Représentation *physique* des données
  - Mise en œuvre concrète des méthodes

Organisation des classes Java

Éléments de base

- Une classe par fichier

Organisation des classes Java

Éléments de base

- Une classe par fichier
- Tout est compris :

Organisation des classes Java

Éléments de base

- Une classe par fichier
- Tout est compris :
  - les spécifications de chaque méthode

Organisation des classes Java

Éléments de base

- Une classe par fichier
- Tout est compris :
  - les spécifications de chaque méthode
  - le codage

Organisation des classes Java

Éléments de base

- Une classe par fichier
- Tout est compris :
  - les spécifications de chaque méthode
  - le codage

Organisation des classes Java

Éléments de base

- Une classe par fichier
- Tout est compris :
  - les spécifications de chaque méthode
  - le codage

Package

- Rassemblement logique de classes

Organisation des classes Java

Éléments de base

- Une classe par fichier
- Tout est compris :
  - les spécifications de chaque méthode
  - le codage

Package

- Rassemblement logique de classes
- Hiérarchie des packages

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

private : les éléments ne sont visibles qu'à partir de la classe

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

private : les éléments ne sont visibles qu'à partir de la classe

protected : les éléments sont visibles des classes dérivées

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

private : les éléments ne sont visibles qu'à partir de la classe

protected : les éléments sont visibles des classes dérivées

par défaut : visibilité package

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

private : les éléments ne sont visibles qu'à partir de la classe

protected : les éléments sont visibles des classes dérivées

par défaut : visibilité package

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

private : les éléments ne sont visibles qu'à partir de la classe

protected : les éléments sont visibles des classes dérivées

par défaut : visibilité package

Persistance

final : l'attribut est initialisé à la construction de l'objet et non modifiable. (Cas des méthodes et des classes)

Protection des constituants d'une classe

Accessibilité

public : les éléments sont visibles par les utilisateurs de la classe

private : les éléments ne sont visibles qu'à partir de la classe

protected : les éléments sont visibles des classes dérivées

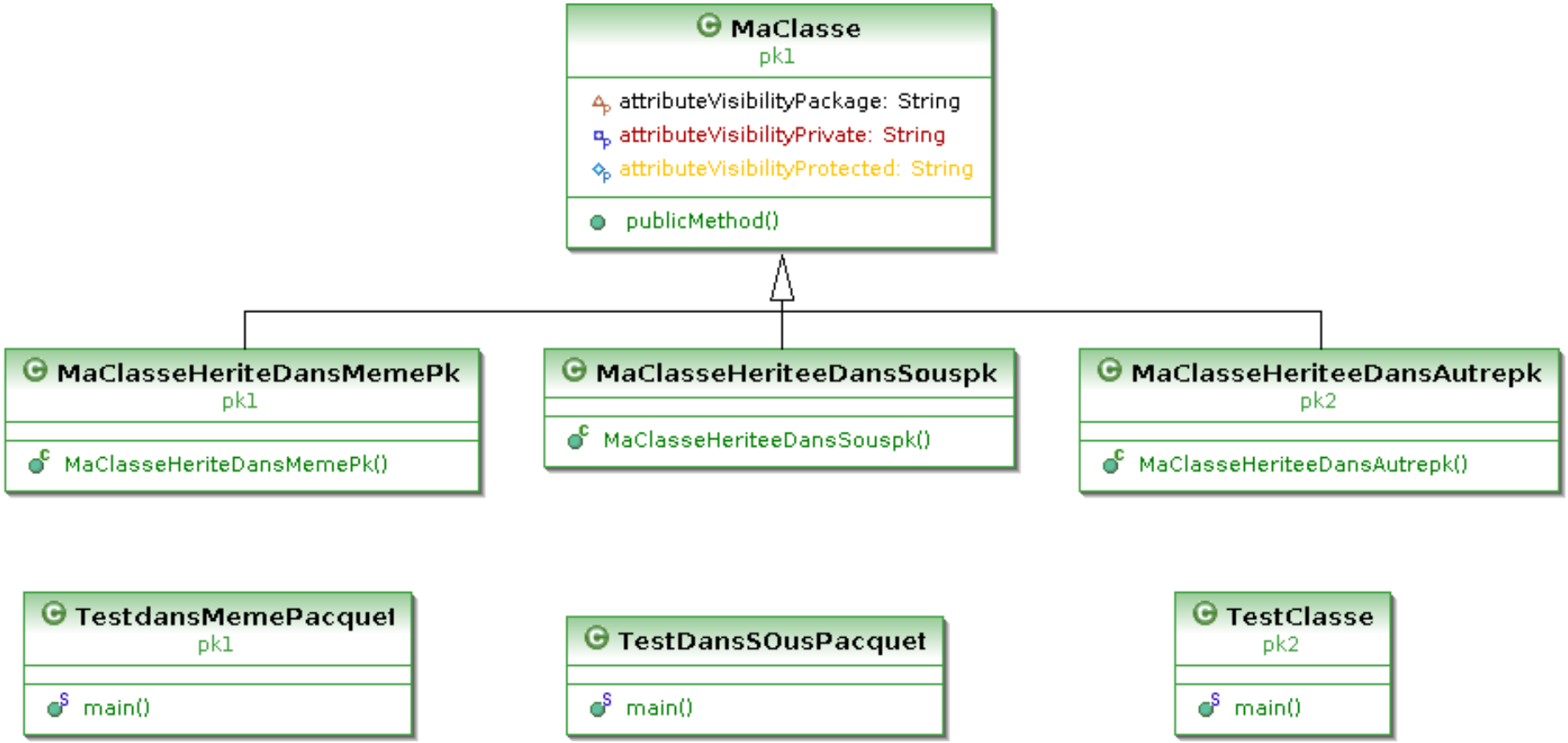
par défaut : visibilité package

Persistance

final : l'attribut est initialisé à la construction de l'objet et non modifiable. (Cas des méthodes et des classes)

static : les éléments sont indépendants des instances

Exemple


MaClasse
pk1
attributeVisibilityPackage: String
attributeVisibilityPrivate: String
attributeVisibilityProtected: String
publicMethod()
MaClasseHeriteDansMemePk
pk1
MaClasseHeriteDansMemePk()
MaClasseHeriteeDansSouspk
MaClasseHeriteeDansSouspk()
MaClasseHeriteeDansAutrepk
pk2
MaClasseHeriteeDansAutrepk()
TestdansMemePacquet
pk1
main()
TestDansSOusPacquet
main()
TestClasse
pk2
main()

Exemple – suite

Code dans les tests

```
1 MaClasse mc = new MaClasse();
2 mc.publicMethod();
3 System.out.println(mc.attributeVisibilityProtected);
4 System.out.println(mc.attributeVisibilityPackage);
5 System.out.println(mc.attributeVisibilityPrivate);
```

numligne	2	3	4	5
même package	OK	OK	OK	KO
sous package	OK	KO	KO	KO
Autre Package	OK	KO	KO	KO

Bonnes pratiques

Nécessité de conventions (Sun)

Code conventions are important to programmers for a number of reasons :

- 80% of the lifetime cost of a piece of software goes to maintenance.

Bonnes pratiques

Nécessité de conventions (Sun)

Code conventions are important to programmers for a number of reasons :

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.

Bonnes pratiques

Nécessité de conventions (Sun)

Code conventions are important to programmers for a number of reasons :

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand -new code more quickly and thoroughly.

Bonnes pratiques

Nécessité de conventions (Sun)

Code conventions are important to programmers for a number of reasons :

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand -new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation
- Normalisation des noms :

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation
- Normalisation des noms :
  - noms significatifs

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation
- Normalisation des noms :
  - noms significatifs
  - packages : `mybeautifulpackage`

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation
- Normalisation des noms :
  - noms significatifs
  - packages : `mybeautifulpackage`
  - classes : `aVeryExplicitClassName`

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation
- Normalisation des noms :
  - noms significatifs
  - packages : `mybeautifulpackage`
  - classes : `aVeryExplicitClassName`
  - variables : `myVariable`

Application au codage

Quelques règles

- Les attributs sont en `private` ou `protected`
- Les méthodes suivant ce que l'on cherche à faire :
  - définir l'interface
  - empêcher l'utilisation
- Normalisation des noms :
  - noms significatifs
  - packages : `mybeautifulpackage`
  - classes : `aVeryExplicitClassName`
  - variables : `myVariable`
  - constantes : `PI`

Cas des méthodes

Méthodes

- Accesseurs :

Cas des méthodes

Méthodes

- Accesseurs :
  - `getAttribute`

Cas des méthodes

Méthodes

- Accesseurs :
  - `getAttribute`
  - `isBooleanAttribute`

Cas des méthodes

Méthodes

- Accesseurs :
  - `getAttribute`
  - `isBooleanAttribute`
  - `setAttribute`

Cas des méthodes

Méthodes

- Accesseurs :
  - `getAttribute`
  - `isBooleanAttribute`
  - `setAttribute`
- validation : `checkState`

Cas des méthodes

Méthodes

- Accesseurs :
  - `getAttribute`
  - `isBooleanAttribute`
  - `setAttribute`
- validation : `checkState`
- ...

Cas des méthodes

Méthodes

- Accesseurs :
  - `getAttribute`
  - `isBooleanAttribute`
  - `setAttribute`
- validation : `checkState`
- ...
- Question : faut-il franciser ?

Javadoc

Un outil de documentation : Javadoc

- Importance de la documentation des méthodes

Javadoc

Un outil de documentation : Javadoc

- Importance de la documentation des méthodes
- Chaque élément **DOIT** être documenté

Javadoc

Un outil de documentation : Javadoc

- Importance de la documentation des méthodes
- Chaque élément **DOIT** être documenté
  - classes, interfaces, ...

Javadoc

Un outil de documentation : Javadoc

- Importance de la documentation des méthodes
- Chaque élément **DOIT** être documenté
  - classes, interfaces, ...
  - constantes, attributs, méthodes

Javadoc

Un outil de documentation : Javadoc

- Importance de la documentation des méthodes
- Chaque élément **DOIT** être documenté
  - classes, interfaces, ...
  - constantes, attributs, méthodes

Javadoc

Un outil de documentation : Javadoc

- Importance de la documentation des méthodes
- Chaque élément **DOIT** être documenté
  - classes, interfaces, ...
  - constantes, attributs, méthodes

```
/**
 * Cette méthode retourne un entier dont la pertinence pour la classe
 * est de première importance
 * @param dd représente une valeur
 * @param ii représente le nombre de ...
 * @return quelquechose qui est proche de 0
 */
public int myDummyMethod(double dd, int ii){
    return 0; }
```

Javadoc

Tags javadoc

@param décrit un paramètre d'une méthode

Javadoc

Tags javadoc

@param décrit un paramètre d'une méthode

@return décrit ce que retourne une fonction

Javadoc

Tags javadoc

@param décrit un paramètre d'une méthode

@return décrit ce que retourne une fonction

@throws exception propagée par la méthode

Javadoc

Tags javadoc

@param	décrit un paramètre d'une méthode
@return	décrit ce que retourne une fonction
@throws	exception propagée par la méthode
@author	auteur de la classe

Javadoc

Tags javadoc

@param	décrit un paramètre d'une méthode
@return	décrit ce que retourne une fonction
@throws	exception propagée par la méthode
@author	auteur de la classe
@version	version de la classe

Javadoc

Tags javadoc

@param	décrit un paramètre d'une méthode
@return	décrit ce que retourne une fonction
@throws	exception propagée par la méthode
@author	auteur de la classe
@version	version de la classe
@see	référence à d'autres méthodes/classes : *@see MyClass#myMethod(prototype)*

Javadoc

Tags javadoc

@param	décrit un paramètre d'une méthode
@return	décrit ce que retourne une fonction
@throws	exception propagée par la méthode
@author	auteur de la classe
@version	version de la classe
@see	référence à d'autres méthodes/classes : *@see MyClass#myMethod(prototype)*
@since	début de validité de la méthode

Javadoc

Tags javadoc

@param	décrit un paramètre d'une méthode
@return	décrit ce que retourne une fonction
@throws	exception propagée par la méthode
@author	auteur de la classe
@version	version de la classe
@see	référence à d'autres méthodes/classes : *@see MyClass#myMethod(prototype)*
@since	début de validité de la méthode
@deprecated	indique la méthode va disparaître dans une version future

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques
     - ***false*** pour les booléens

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques
     - ***false*** pour les booléens
     - ***null*** pour les objets

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques
     - ***false*** pour les booléens
     - ***null*** pour les objets
     - par la valeur éventuellement indiquée lors de la déclaration
       private int *myInteger = 45;*

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques
     - ***false*** pour les booléens
     - ***null*** pour les objets
     - par la valeur éventuellement indiquée lors de la déclaration
       private int *myInteger = 45;*
  2. À éviter ! Exécution du code *libre*

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques
     - ***false*** pour les booléens
     - ***null*** pour les objets
     - par la valeur éventuellement indiquée lors de la déclaration ***private int*** *myInteger = 45;*
  2. À éviter ! Exécution du code *libre*
  3. Exécution du code spécifique

Constructeurs

Principe

- Initialisation des attributs avec des valeurs pertinentes
- Possibilité de surcharge
- Fonctionnement :
  1. Initialisation des attributs par :
     - 0 pour les numériques
     - ***false*** pour les booléens
     - ***null*** pour les objets
     - par la valeur éventuellement indiquée lors de la déclaration ***private int*** *myInteger = 45;*
  2. À éviter ! Exécution du code *libre*
  3. Exécution du code spécifique
- Si aucun n'est défini, constructeur par défaut

Déclaration et initialisation

Déclaration

- Comme pour les autres *types* :
  MyClass myObjectOfMyClass

Déclaration et initialisation

Déclaration

- Comme pour les autres *types* :
  MyClass myObjectOfMyClass
- Définition d'une référence

Déclaration et initialisation

Déclaration

- Comme pour les autres *types* :
  MyClass myObjectOfMyClass
- Définition d'une référence
- Pas d'initialisation à cet instant

Déclaration et initialisation

Déclaration

- Comme pour les autres *types* :
  MyClass myObjectOfMyClass
- Définition d'une référence
- Pas d'initialisation à cet instant

Déclaration et initialisation

Déclaration

- Comme pour les autres *types* :
  MyClass myObjectOfMyClass
- Définition d'une référence
- Pas d'initialisation à cet instant

Initialisation

- Par l'appel d'un constructeur :
  myObject = ***new*** *myClass([params]);*

Déclaration et initialisation

Déclaration

- Comme pour les autres *types* :
  MyClass myObjectOfMyClass
- Définition d'une référence
- Pas d'initialisation à cet instant

Initialisation

- Par l'appel d'un constructeur :
  myObject = ***new*** *myClass([params]);*
- Par l'affectation d'un autre objet

Déclaration et initialisation

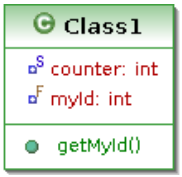
Déclaration

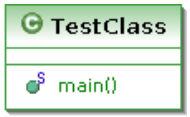
- Comme pour les autres *types* :
  MyClass myObjectOfMyClass
- Définition d'une référence
- Pas d'initialisation à cet instant

Initialisation

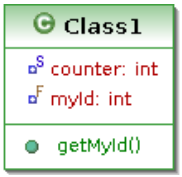
- Par l'appel d'un constructeur :
  myObject = ***new*** *myClass([params]);*
- Par l'affectation d'un autre objet
- Par l'affectation de la référence ***null***

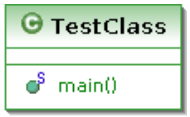
Exemple





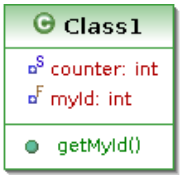
Exemple

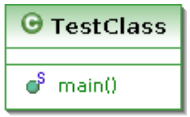





```
public class Class1 {
  private static int counter = 0;
  private final int myId = counter++;
  public int getMyId() {
          return myId;}
}
```

Exemple






```
public class Class1 {
  private static int counter = 0;
  private final int myId = counter++;
  public int getMyId() {
          return myId;}
}
```

```
Class1 c1 = new Class1();
System.out.println(c1.getMyId());
Class1 c2 = c1;
System.out.println(c2.getMyId());
Class1 c3 = new Class1();
System.out.println(c3.getMyId());
```

Tableaux

Définition

- *MaClasse monTab[];*

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Initialisation

- *monTab = **new** MaClasse [10];*

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Initialisation

- *monTab = **new** MaClasse [10];*
- *monTab = **null**;*

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Initialisation

- *monTab =* ***new*** *MaClasse [10];*
- *monTab =* ***null****;*
- ou :

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Initialisation

- *monTab =* ***new*** *MaClasse [10];*
- *monTab =* ***null****;*
- ou :

Tableaux

Définition

- *MaClasse monTab[];*
- *MaClasse [] monTab*

Initialisation

- *monTab =* ***new*** *MaClasse [10];*
- *monTab =* ***null****;*
- ou :

```
int [] monTabInt = new int[] {1, 2, 3, 4, 5};
monTabInt = new int[15];
for(int i= 0; i<15;i++)
    monTabInt[i]= i;
```

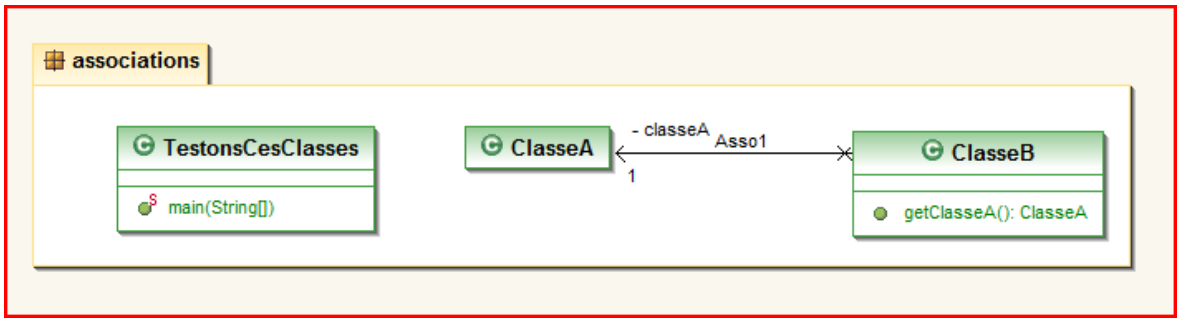
Tableaux bi-dimensionnels

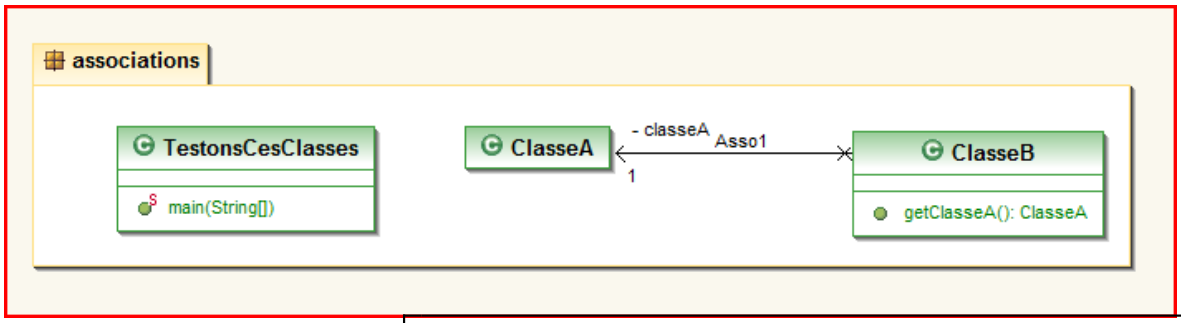
```
int [][] tabCarre = new int [4][4];

    int [][] tabTriangulaire = new int [10] [];
    for(int i = 0; i< tabTriangulaire.length; i++)
            tabTriangulaire[i] = new int [i+1];

    int[] temp;
    temp = tabTriangulaire[0];
    tabTriangulaire[0] = tabTriangulaire[9];
    tabTriangulaire[9] = temp;

    System.out.println(tabTriangulaire[0].length);
    System.out.println(tabTriangulaire[9].length);
```


associations
TestonsCesClasses
main(String[])
ClasseA
- classeA
Asso1
1
ClasseB
getClasseA(): ClasseA



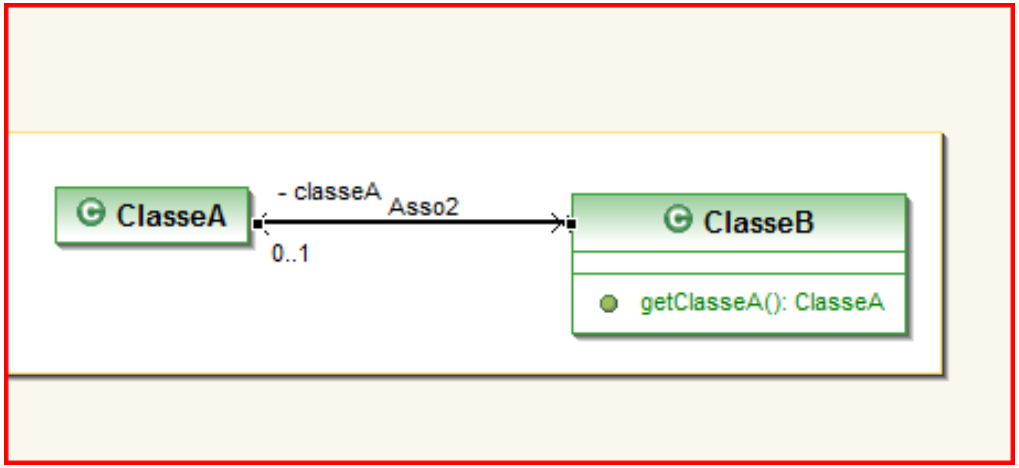

```
package associations;


public class ClasseA {

}
```

```
package associations;
public class ClasseB {
   private ClasseA classeA = new
        ClasseA();
      /**
       * Getter of the property
       * @return Returns the classeA.c
       */
      public ClasseA getClasseA() {
          return classeA;
}}
```

Autres liens



Autres liens