IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP1 – La notion de producteur

Gestion de stock

Méthodes et programmation objet – PC-DI-MPO

Objectifs

L'objectif de cette séance est de débuter la programmation Java et de pratiquer les concepts basiques de la programmation objet.

Contexte

Le contexte est identique à celui de la séance précédente. Après avoir défini les classes `Product` et `Stock`, nous allons définir et réaliser la classe `Producer`.

Un producteur fabrique des produits et les range dans un stock unique. À un instant donné, le producteur ne connaît que le produit qu'il est en train de créer. Un stock contient de nombreux produits (qui pourraient être produits par plusieurs producteurs). Chaque produit est fabriqué par un seul producteur et est entreposé dans un seul stock.

Les classes `Product` et `Stock` vues dans la séance précédente sont accessibles depuis Moodle `https://moodle.imt-atlantique.fr/course/view.php?id=19&section=1`.

Préliminaires

Lors des travaux pratiques de l'UE MPO, vous utiliserez l'environnement de développement Eclipse.

En cas de problème ou pour plus d'information, reportez-vous à l'utilisation d'Eclipse figurant en annexe A du sujet de TP.

Une présentation détaillée de l'utilisation d'Eclipse est également disponible sur Moodle `https://moodle.imt-atlantique.fr/course/view.php?id=19&section=1` (onglet Ressources).

Une première classe

Exercice 1 (*Producer*)

La classe `Producer` crée des objets de type `Product` et les range dans un objet de type `Stock`. Un producteur possède un nom (`name`) qui lui est attribué à sa création, il connaît aussi à sa création l'objet de type `Stock` (`myStock`) où il doit ranger ses produits.

- **Variables d'instance :**
  - `name` : son nom
  - `myStock` : le stock où il dépose ses produits
- **Constructeurs :**
  - un constructeur avec comme paramètres son nom et le stock qu'il utilise
- **Méthodes d'instance :**
  - `String getName()` : donne le nom du producteur
  - `void setName(String name)` : change le nom du producteur
  - `Stock getStock()` : donne le stock utilisé par le producteur
  - `void produce(String productName)` : création d'un nouveau produit dont le nom est passé en argument et rangement dans le stock
  - `String toString()` : rend une chaîne de caractères décrivant le producteur
- **Test :**
  - une méthode `main` qui teste cette classe

▷ **Question 1.1 :**
Récupérer l'archive du projet contenant les classes vues en TD sous Moodle `https://moodle.imt-atlantique.fr/course/view.php?id=19&section=1`.

▷ **Question 1.2 :**
Lancer Eclipse (menu Application/Programmation/Eclipse). Importer l'archive précédemment téléchargée comme projet existant (cf annexe A). Ouvrir les classes `Product` et `Stock` du projet puis les exécuter.

▷ **Question 1.3 :**
Réaliser la classe `Producer` en ignorant les problèmes d'erreur (ajout d'un produit dans un stock plein ou de retrait d'un produit d'un stock vide).

Votre classe doit bien sûr être *bien* testée.

Documentation de programmes Java

Pour produire la documentation d'une classe, le JDK (*« Java Development Kit »*) inclut l'outil `javadoc`. À partir du fichier source de la classe, il construit une page Web décrivant les différentes propriétés de la classe. Pour le réaliser, il utilise les commentaires commençant par `/**` et se terminant par `*/`. Vous trouverez ci-dessous un exemple de classe (la classe `Complex`) contenant la documentation en format `javadoc`.

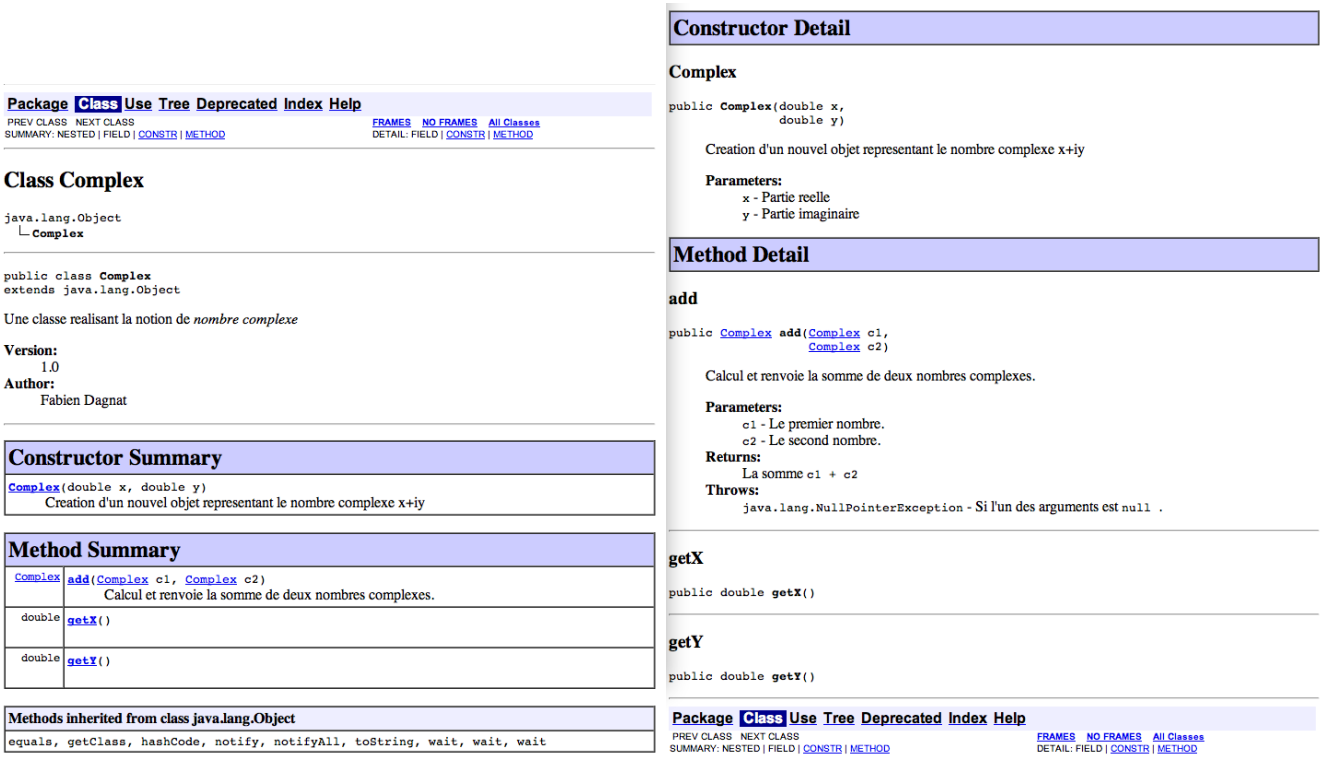
```
/**
 * Une classe realisant la notion de <i>nombre complexe</i>
 *
 * @author Fabien Dagnat
```

```
 * @version 1.0
 */
public class Complex {
  private final double x;
  private final double y;

  /**
   * Creation d'un nouvel objet representant le nombre complexe x+iy
   *
   * @param x Partie reelle
   * @param y Partie imaginaire
   */
  public Complex(double x, double y) {
    this.x = x;
    this.y = y;
  }

  /**
   * Calcul et renvoie la somme de deux nombres complexes.
   *
   * @param c1 Le premier nombre.
   * @param c2 Le second nombre.
   * @return La somme <code> c1 + c2 </code>
   * @exception NullPointerException Si l'un des arguments est <code> null </code>.
   */
  public Complex add(Complex c1, Complex c2) {
    return new Complex(c1.getX() + c2.getX(), c1.getY() + c2.getY());
  }

  public double getX() {
    return this.x;
  }

  public double getY() {
    return this.y;
  }
}
```

L'outil `javadoc` peut être appelé en exécutant la commande `javadoc`. Par exemple, la commande suivante génère la documentation (on dit aussi API) de la classe `Complex` dans le répertoire `doc`.

```
javadoc -d doc Complex.java
```

La documentation produite est un ensemble de pages HTML dont l'aperçu suit :



Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Complex

java.lang.Object
└ **Complex**

public class **Complex**
extends java.lang.Object

Une classe realisant la notion de *nombre complexe*

Version: 1.0
Author: Fabien Dagnat

Constructor Summary

Complex(double x, double y)	Creation d'un nouvel objet representant le nombre complexe x+iy

Method Summary

Complex	**add**(Complex c1, Complex c2) Calcul et renvoie la somme de deux nombres complexes.
double	**getX**()
double	**getY**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Complex

public **Complex**(double x, double y)

Creation d'un nouvel objet representant le nombre complexe x+iy

Parameters: x - Partie reelle, y - Partie imaginaire

Method Detail

add

public Complex **add**(Complex c1, Complex c2)

Calcul et renvoie la somme de deux nombres complexes.

Parameters: c1 - Le premier nombre. c2 - Le second nombre.
Returns: La somme c1 + c2
Throws: java.lang.NullPointerException - Si l'un des arguments est null .

getX

public double **getX**()

getY

public double **getY**()

Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Avec Eclipse

L'outil Eclipse intègre la génération de la documentation javadoc de vos classes. Il suffit, pour cela, d'utiliser le menu Project – Generate Javadoc. Une fenêtre de dialogue apparaît et vous demande de sélectionner les classes à documenter. Sélectionner la classe `Complex` puis Finish. Un répertoire `doc` est créé dans votre projet. Ouvrir le fichier `Complex.html` présent dans ce répertoire.

Exercice 2 (*Documentation*)

▷ **Compléter la classe `Producer` en lui ajoutant tous les commentaires javadoc nécessaires. Puis générer les pages HTML de documentation.**

Annexes

A Utilisation d'Eclipse

Dans le cadre de ce module nous utiliserons l'environnement de développement Eclipse dont la fenêtre principale est présentée dans la figure 1. Il s'agit d'un logiciel comprenant entre autres les éléments suivants nécessaires au développement d'applications informatiques :

— un éditeur de texte capable de s'adapter à la syntaxe de plusieurs langages,

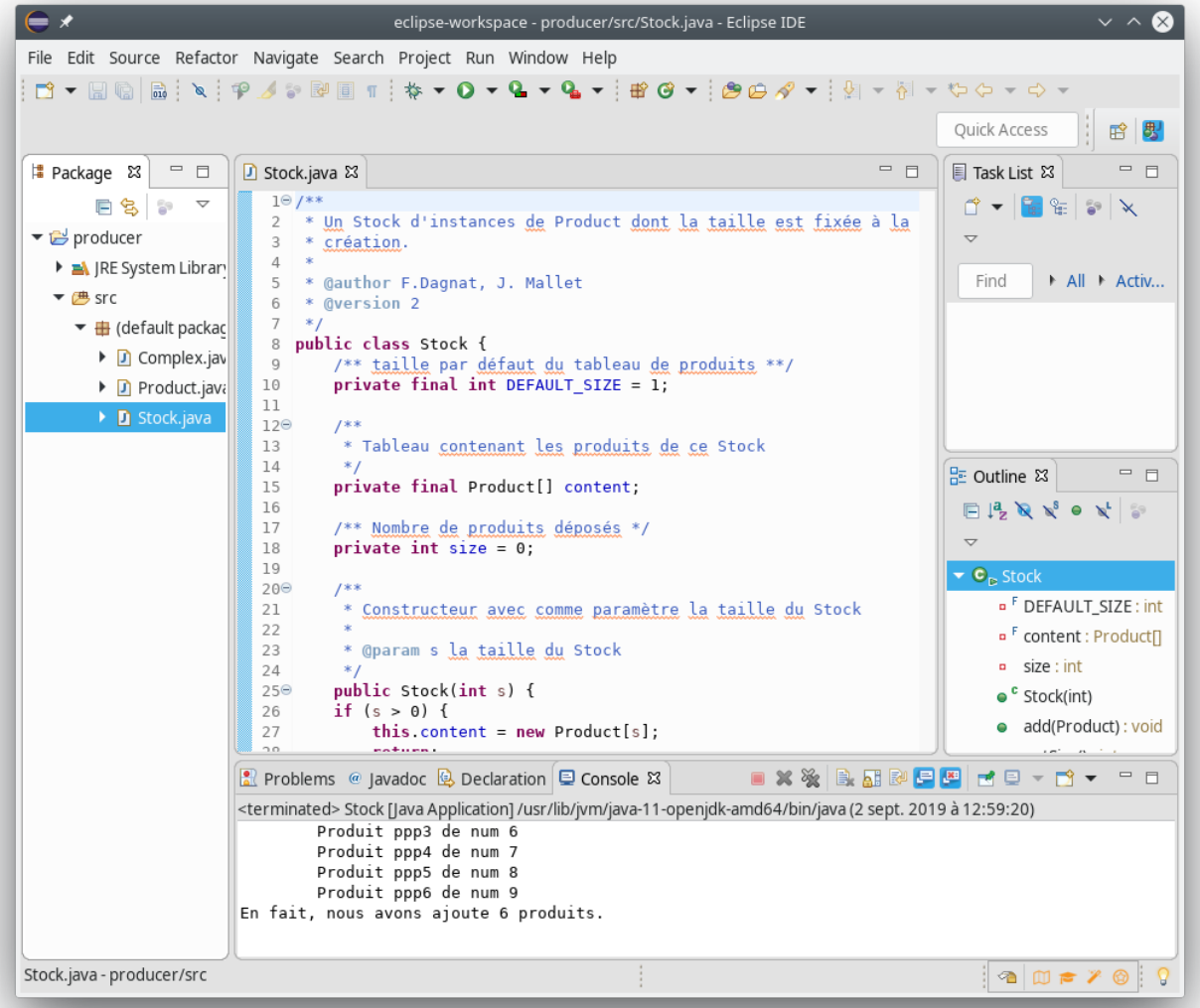


FIGURE 1 – La fenêtre principale d'Eclipse

— des liens vers les compilateurs,
— des commandes d'exécution.

Le navigateur permet de présenter plusieurs vues du système. On l'utilise pour chercher un fichier existant (classe, diagramme, ...) ou pour créer un nouveau fichier. On navigue dans ces vues par des arborescences qu'on plie ou déplie.

L'éditeur de texte permet de créer des entités en fonction de la nature du fichier. Pour créer une entité on utilise les menus ou la barre d'icônes. Pour visualiser un fichier existant, on double-clique sur le fichier considéré dans le navigateur et il apparaît dans l'éditeur.

Lancer eclipse

Pour lancer Eclipse, utiliser le menu Application/Programmation/Eclipse. Vous pouvez également, dans un terminal, exécuter la commande `eclipse`.

Une fenêtre vous demande le répertoire qui contiendra les projets (*workspace*). Pour choisir le répertoire par défaut, cliquer sur Launch. Puis fermer l'onglet Welcome.

Importer un projet existant

Lorsqu'un projet Eclipse vous est fourni sous forme d'archive :

1. récupérer le projet Eclipse ;
2. importer le projet (File – Import puis General – Existing Projects into Workspace) puis Next puis sélectionner Select archive file. En utilisant Browse, choisir l'archive contenant le projet.

Compilation.

Avec Eclipse il n'est pas nécessaire de compiler les classes écrites. La compilation est lancée automatiquement. Par défaut, les classes compilées se trouvent dans le répertoire `bin` du projet contenant l'application.

Exécution.

Pour lancer l'exécution d'une application, sélectionner le projet puis, à l'aide du menu contextuel, Run As – 1 Java Application.

Feedback.

Remarquez qu'Eclipse vous montre le résultat des commandes exécutées ainsi que de l'exécution de vos classes dans la partie inférieure de la fenêtre, dans l'onglet Console.

Navigation dans les sources.

Sur un appel de méthode, Ctrl + click droit sur la souris – Open declaration vous positionne sur la définition de la méthode.

Complétion du nom des méthodes et attributs d'une classe.

Sur un appel de méthode ou l'accès à un attribut d'une classe, Ctrl + espace vous propose les complétions possibles. Il vous suffit de choisir celle souhaitée.

B Compilation et exécution en ligne de commande

Pour compiler une classe java (un fichier `Nom.java`), il faut utiliser la commande `javac`. Comme toute commande Linux, il est possible de compiler plusieurs fichiers de différents répertoires simultanément. L'option `-d` permet de spécifier un répertoire destination pour les fichiers de bytecode (les fichiers `Nom.class`). Par exemple :

```
javac -d build Test*.java src/truc/*.java src/machin/Logo.java
```

Il est possible de lancer l'exécution (l'interprétation du bytecode) par la JVM d'une classe qui contient une méthode `main`[1] par la commande `java`. Par exemple :

```
java monpaquet.Tutu
```

1. À condition qu'elle soit **`public static`**, qu'elle n'ait pas de résultat (**`void`**) et qu'elle prenne en paramètre un tableau de chaînes de caractères (**`String[]`**).