

Java : Partie 2

Les Bases du langage

Juin 2013

Auteur : Chouaïb LAGHLAM

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Bases du Langage Java

Préalabe 5

Notion de Workspace dans Eclipse	5
Quand crée – t –on un Workspace ?	8
Changer de Workspace	8
Notion de projet dans Eclipse	9
Projet java à utiliser ici	10
Arborescence d'un projet java de type «Java Project»	14
Notion de package	15
Notion de classe	17
Notion de Méthode	21
Méthode main	21
Projet et application	21
Tester le projet.....	22
Déployer l'application	22
Eléments du langage	23
Premier code en java	23
Commentaires.....	27
Commentaire sur une seule ligne	27
Commentaire sur plusieurs lignes	27
Syntaxe d'une instruction élémentaire en Java.....	27
Une instruction élémentaire.....	27
Une instruction bloc.....	27
Une instruction hors-piste	28
Afficher des données sur la console	28
Erreurs de saisie	28
Aide à la saisie d'instructions.....	30
Java aime la casse	30
Variables de type primitif	30
Variable	31

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Nom de variable.....	31
Type de variable.....	31
Types primitifs.....	31
Le type String	32
Déclaration d'une variable.....	32
Initialisation d'une variable	33
Affectation d'une variable	33
Portée d'une variable.....	33
Déclarer une constante.....	36
Une instruction java est une fonction qui vient de quelque part.....	37
Demander des données à l'utilisateur	38
Classe scanner.....	38
Réaliser des tests	39
If	39
Tester plusieurs conditions	41
Opérateurs de comparaison	42
Opérateurs logiques.....	42
Switch.....	43
Réaliser des boucles.....	45
For	45
While	47
Do while	48
Opérateurs de calcul.....	49
Opérateurs d'assignation.....	50
Opérateurs d'incrémentement	50
Manipuler des chaînes de caractères	50
Conversion de types	52
Utiliser des tableaux	53
Gestion des exceptions	56
Exemple d'erreurs à l'exécution	56
Les exceptions.....	57
Gérer les Exceptions	57

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Comment gérer les Exceptions ?	57
Les méthodes de classes	58
Syntaxe générale d'une méthode.....	58
Signature d'une méthode	59
Exemples de signatures de méthodes	59
Appels de méthodes : Exemples en java	60
Une astuce pour organiser les méthodes	61
Débogage de l'application	62
Placer de points d'arrêts.....	62
Lancer le débogage	64

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Préalabe

Le monde de développement en Java est un monde très riche en outils, en technologie, en architectures, ...etc. qu'il est impossible de faire un support de cours qui couvrirait tout : j'arriverais peut être à 5000 pages sans avoir tout dit et les choses dites seront à mettre à jour dans 3 ou 4 ans.

Le cours que je vous propose sur « Java » est décomposé volontairement en plusieurs parties : «les bases du langage», « Programmation Orientée Objet en Java », « Accès aux BD en Java», « client riche en Java », « Programmation Web », « Java 2E »,Etc.

Ce découpage facilitera l'apprentissage. Chacun selon, son niveau, ira voir la partie qui l'intéresse le plus.

Pour le débutant, il est recommandé de lire les différentes parties dans l'ordre que j'ai donné à ces parties.

Sachez également que je cherche dans chaque partie à vous ouvrir la voie pour comprendre le thème abordé :

Donc l'accent est mis sur la pédagogie : explications + exemples + exercices corrigés.

Je ne cherche pas à être exhaustif sur un thème. Par exemple : si vous avez compris le principe des «Servlets» avec exemples et exercices à l'appui : le but est atteint. Je vous fais confiance pour être capable d'approfondir les cas particuliers que vous rencontrerez sur votre chemin professionnel.

Quelques sites internet pour s'informer sur Java :

Site officiel de java : <http://docs.oracle.com/javase/7/docs/api/>

Sites des développeurs : <http://fr.openclassrooms.com/>
<http://www.developpez.com/>
<http://www.developpez.net/forums/>

Merci de m'adresser toute erreur ou remarque ou question sur mon email : « chouaib.laghlam@laposte.net » en me précisant la partie concernée, le thème abordée, le numéro de page et l'erreur éventuelle.

Notion de Workspace dans Eclipse

Un **Workspace** (ou espace de travail) est tout simplement un dossier Windows. C'est le dossier principal dans lequel seront Stockés vos projets sous Eclipse.

Eclipse stockera, dans le Workspace, des informations sur vos projets,

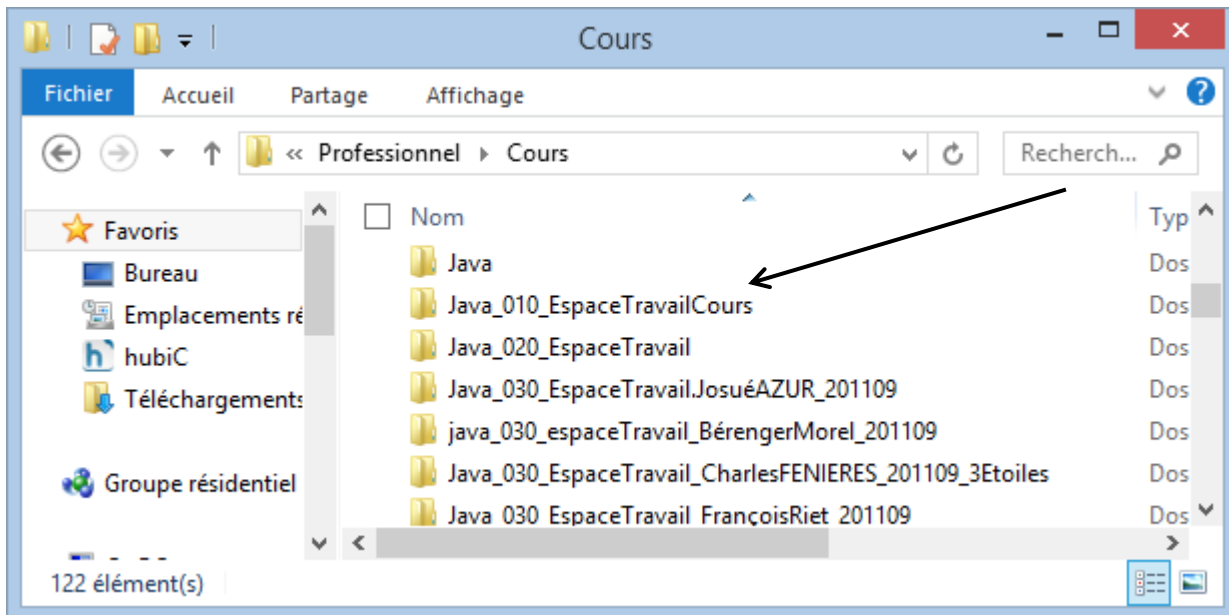
Nous pouvons créer autant de Workspaces que souhaités pour des raisons d'organisation.

A FAIRE : Créez un dossier sous Windows et appelez-le «**Java_010_EspaceTravailCours**» :

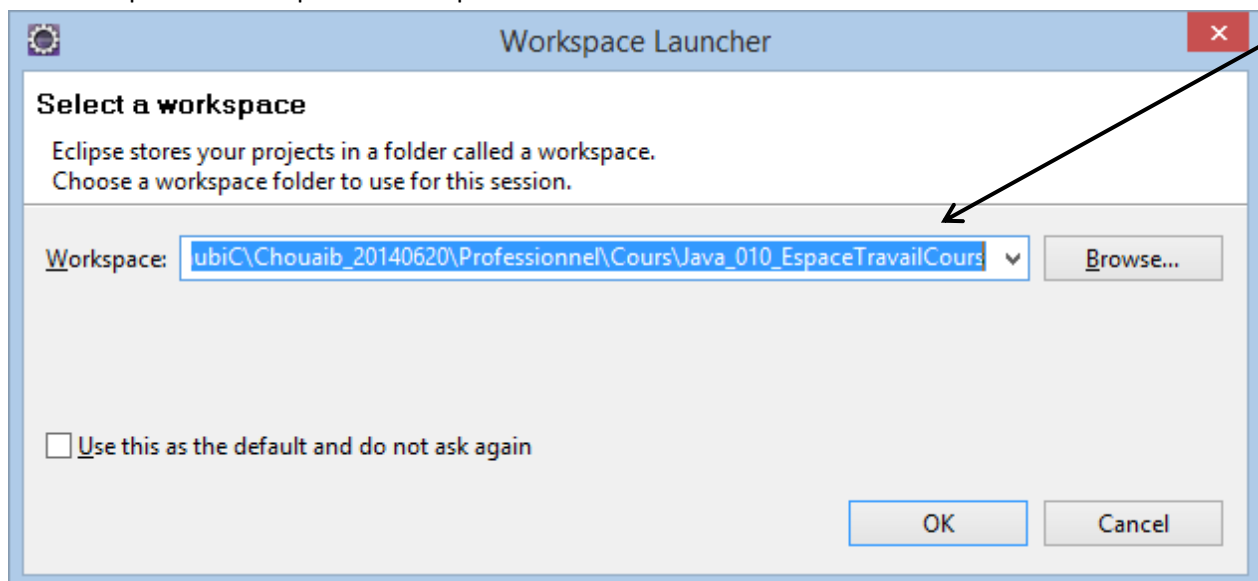
Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Lancez Eclipse et lui indiquez le Workspace :

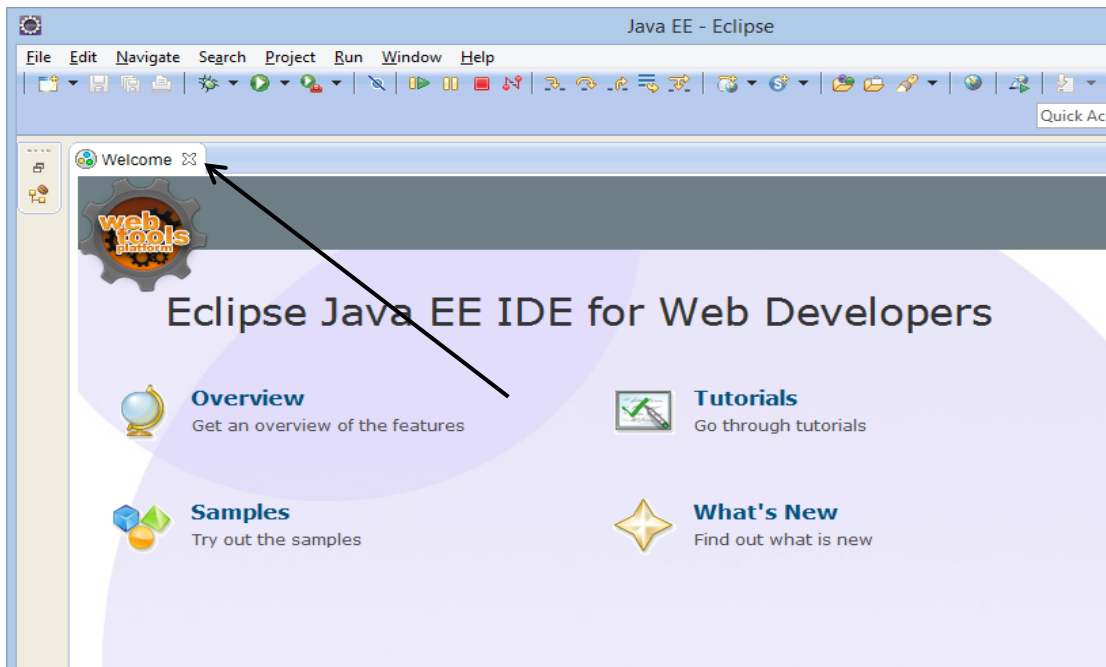


Cliquez sur «OK» : Eclipse s'ouvre.

Concepteur Développeur Informatique

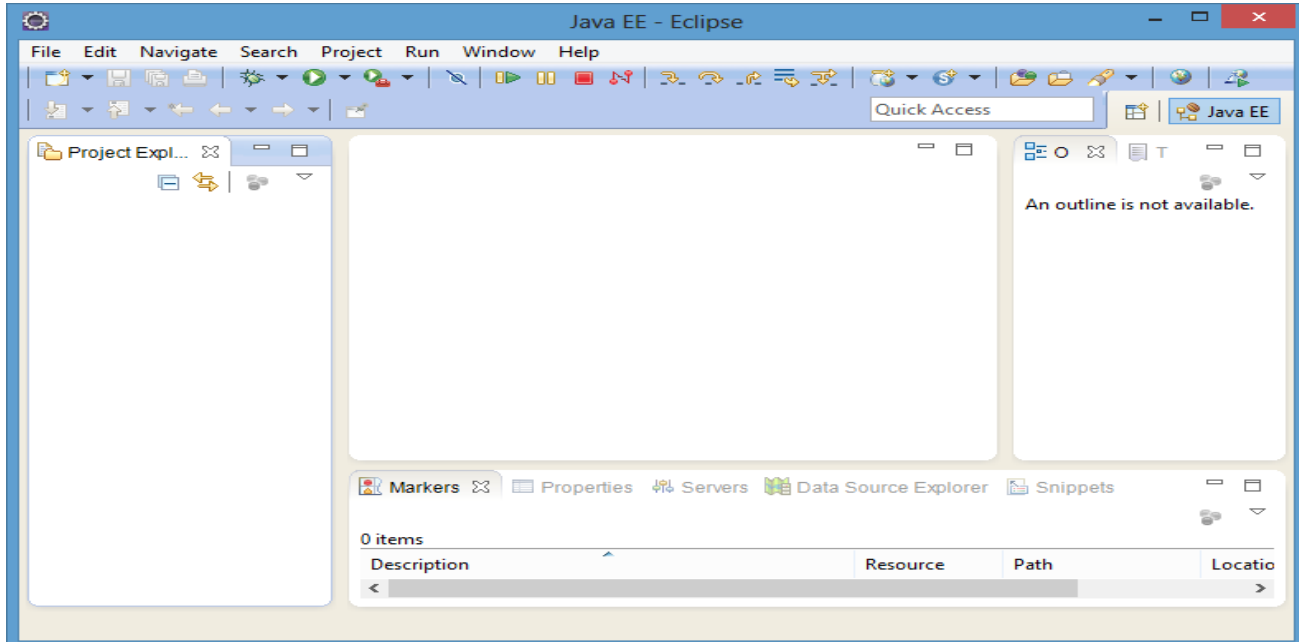
Module : Programmation Java

Bases du Langage



Au premier lancement d'Eclipse : une fenêtre d'accueil s'affiche au milieu, comme ci-dessus : il suffit de cliquer sur la croix « X » de l'onglet « Welcome » pour la fermer.

Vous avez la fenêtre ci-dessous :



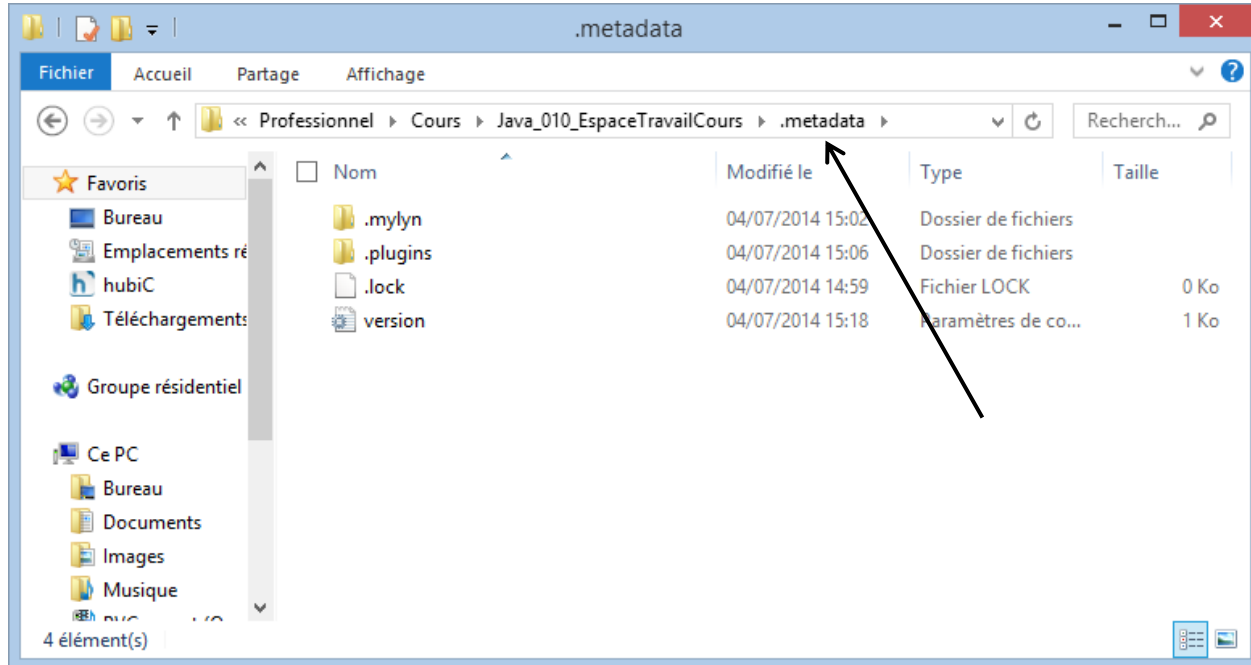
Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Si vous allez voir, par curiosité, dans l'explorateur de Windows le dossier qui sert de Workspace :

Vous trouverez un sous-dossier créé par Eclipse : « **.metadata** » qui contiendra des informations sur les futurs projets que vous allez créer sous Eclipse.



Vous reviendrez plus tard réexaminer ce dossier : il contiendra des sous-dossiers : chacun correspondant à un projet.

Quand crée – t –on un Workspace ?

Nous créons au moins un Workspace pour travailler sous Eclipse. On crée à l'intérieur de celui-ci des applications (on les appellera plus loin des projets).

Habituellement les chefs de projets créent un Workspace par domaine métier.

Par exemple, dans une Banque, on pourrait avoir ceci :

Un Workspace pour le domaine de la gestion des clients qui contiendra plusieurs futures applications (plusieurs projets) :

- Gestion des clients particuliers,
- Gestion des clients Entreprises,
- Gestion des prêts immobiliers,

Un Workspace pour le domaine des titres boursiers :

- Gestion des Titres standards,
- Gestion de.....

Un autre chef de projet créera, peut-être, un Workspace par future application.

Eclipse n'impose pas une **organisation fonctionnelle** particulière.

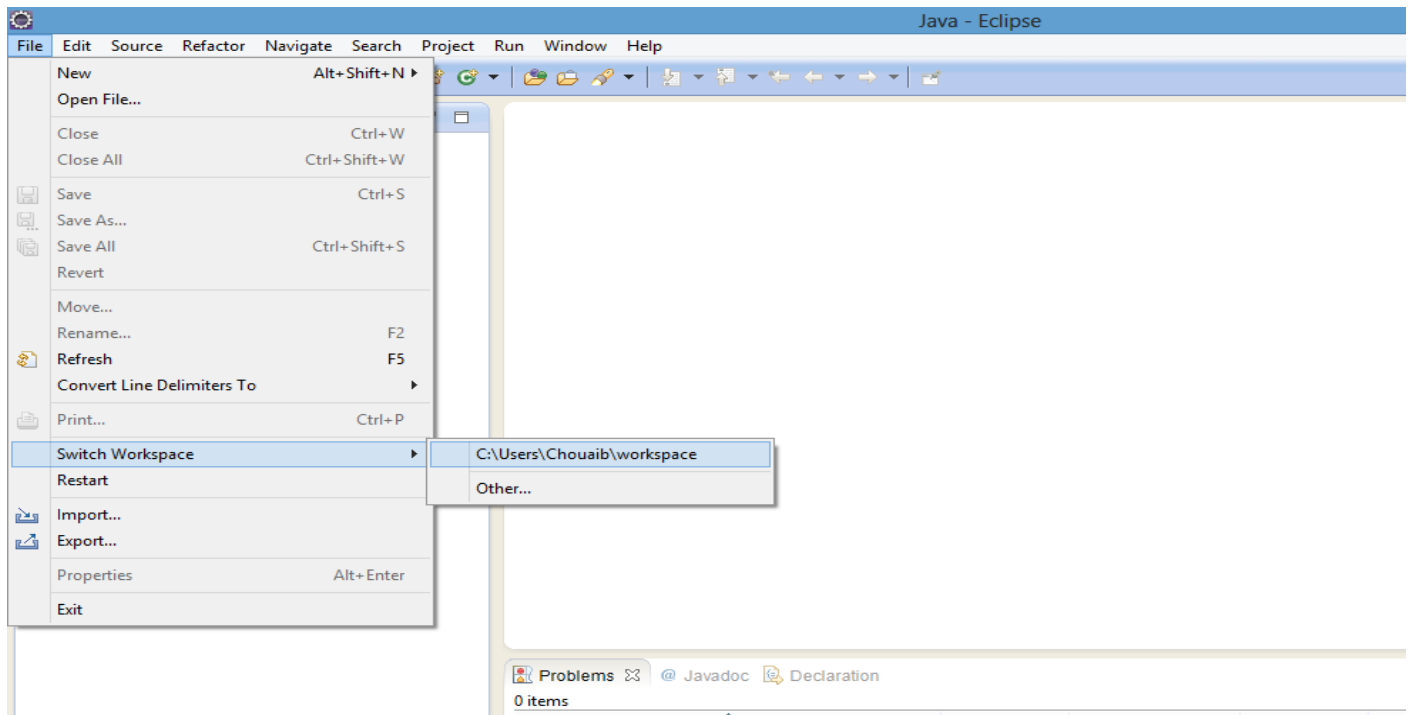
Changer de Workspace

Plus tard si vous créez plusieurs Workspaces avec, dans chacun des projets, vous pourrez passer d'un Workspace à un autre ainsi :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Lors d'un changement de Workspace, Eclipse redémarre.

Notion de projet dans Eclipse

Un projet est une application logicielle en cours de fabrication. Une application n'est plus un projet lorsqu'elle est déployée définitivement auprès des utilisateurs.

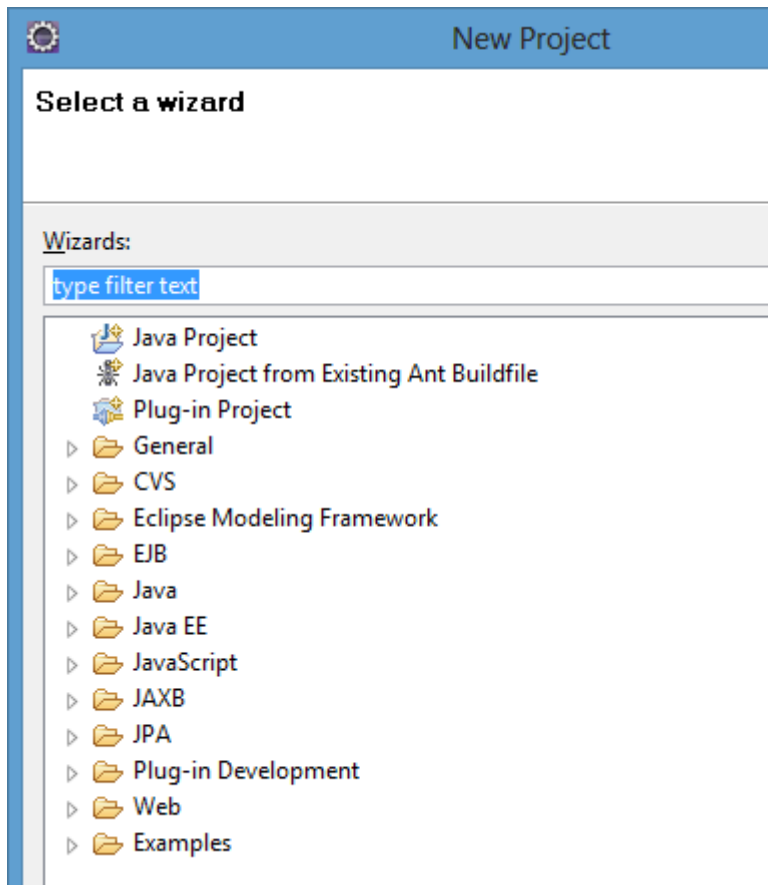
Selon le type d'application à fabriquer (logiciel fenêtré, application pour mobile, montage vidéo, site Web, conception UML,), et selon ce que vous avez ajouté à Eclipse (les plugins), vous devez choisir un type parmi ceux proposés par Eclipse :

Exemple de types de projets proposés par Eclipse :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Projet java à utiliser ici

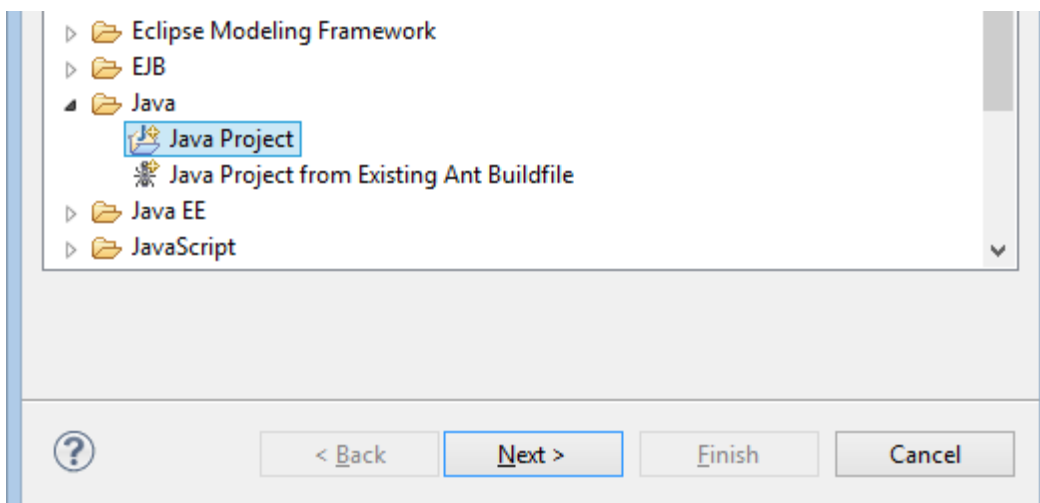
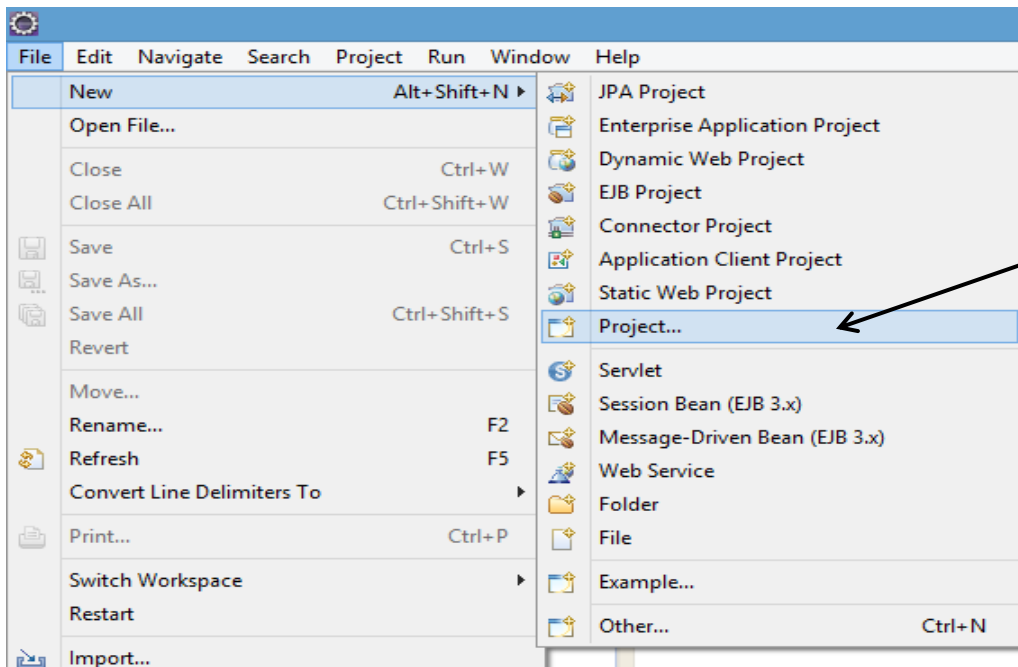
Dans notre apprentissage des « Bases du Langage java », nous allons choisir le type de projet « Java Project » :

A FAIRE : Une fois Eclipse ouvert dans le Workspace «**Java_010_EspaceTravailCours**» ,
Demandez la création d'un projet de type « **Java Project** » :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Nommez le projet : «**prj_Java_010_LesBases**», choisissez la version de la machine virtuelle visée (vous pensez au déploiement future de l'application : sur quelle machine sera elle installée ?) :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

New Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jre8')

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

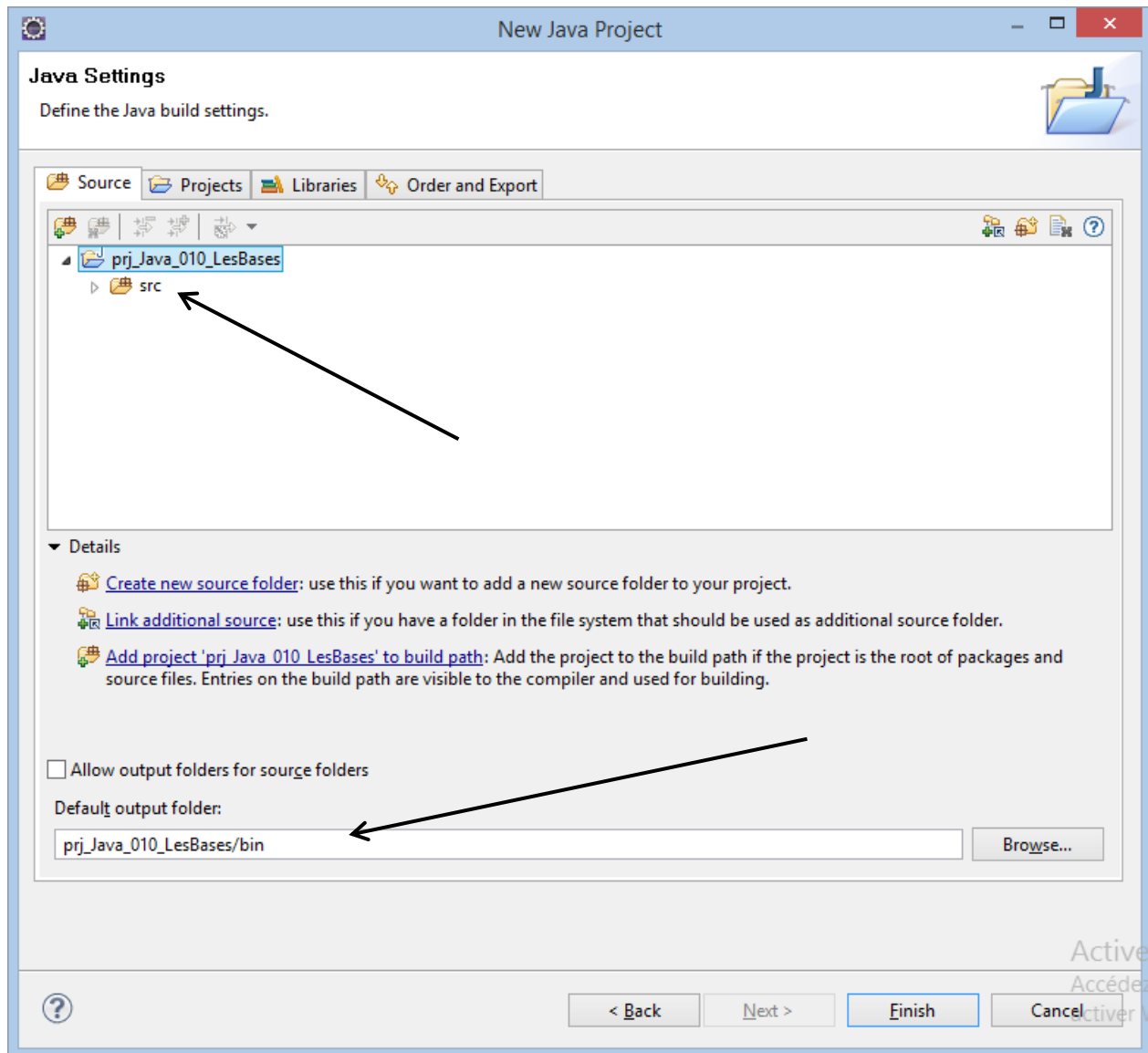
i The default compiler compliance level for the current workspace is 1.4. The new project will use a project compliance level of 1.7.

Laissez les autres choix par défaut et cliquez sur « Next » :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Eclipse nous indique ci-dessus, qu'il va créer un sous-dossier «**src**» pour ranger dedans vos futurs codes sources en java, et un sous-dossier «**bin**» dans lequel Éclipse mettre les versions binaires (Byte code) de nos sources.

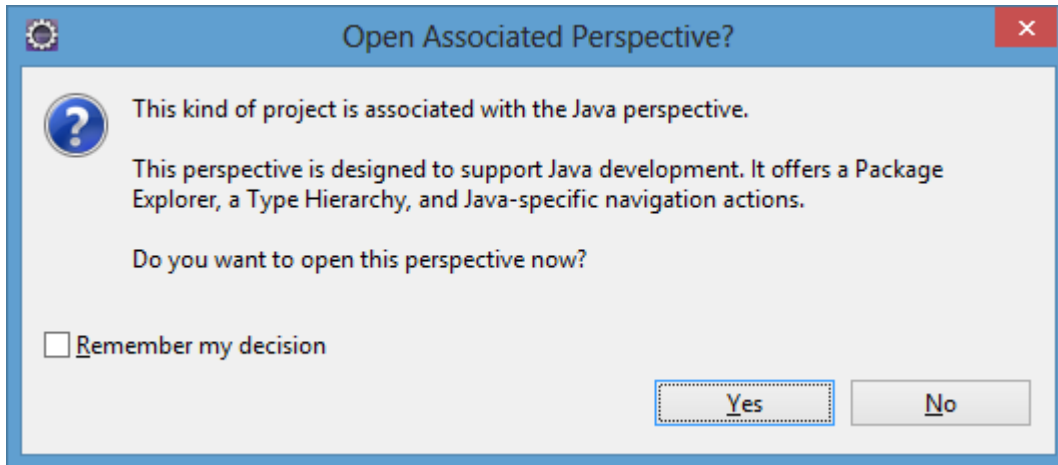
Cliquez sur «**Finish**»

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Eclipse vous informe, dans une dernière fenêtre, qu'il va changer de perspective : c'est-à-dire qu'il va arranger Les différentes fenêtres d'Eclipse en fonction du nouveau type de projet que vous venez de créer :

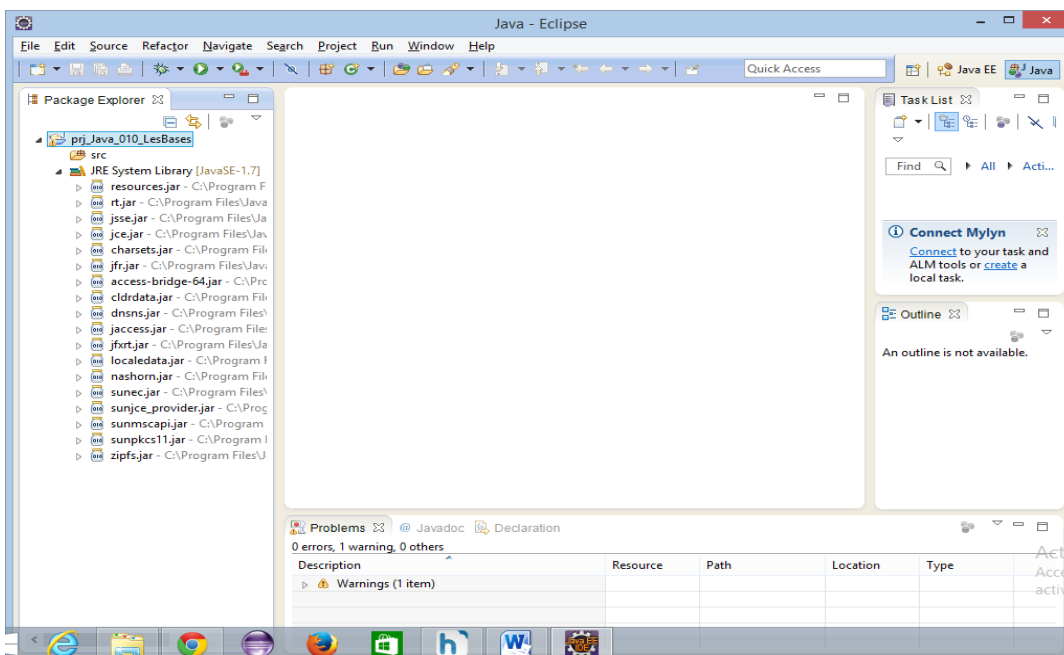


Cliquez sur « **Yes** ».

Votre projet est ouvert.

Arborescence d'un projet java de type «Java Project»

Selon le type de projet Java que vous créez (un Java Projet ou un Dynamic Web Project ou un EJB Project ou), Éclipse Vous imposera une arborescence de dossiers, sous-dossiers, fichiers, Différente.



Ci-dessus, à gauche, notre projet se compose d'un sous dossier «src» et d'une espèce d'étagère nommée « **JRE System Library** » contenant plusieurs fichiers ayant pour extension « .jar »

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Cette étagère est en réalité un ensemble d'exécutables, écrits eux-mêmes en java, et nécessaires pour programmer en java.

L'étagère s'appelle réellement une bibliothèque (ou **Library**) et les exécutables avec .jar se sont des **archives java** (JAR = Java **AR**chive).

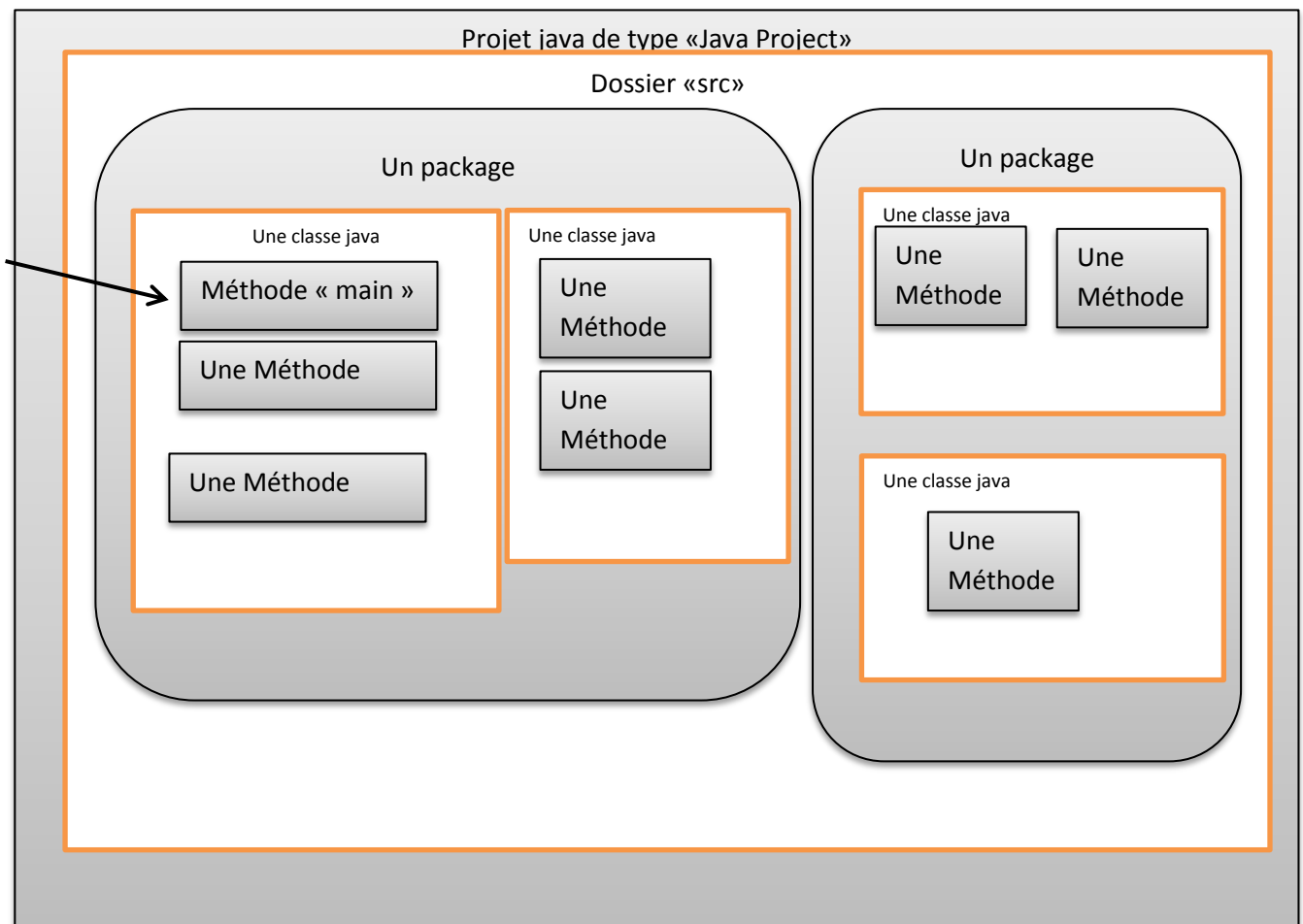
Lorsque vous coderez votre première application : Lorsqu'elle est bien terminée et prête à être utilisée par les autres : vous générerez une archive java (un fichier .jar) pour le fournir aux futurs utilisateurs (c'est l'équivalent d'un exécutable dans d'autres technologies),

L'arborescence que l'on vient de voir est assez simple car le type de projet choisi (Java Project) est simple.

Nous allons bientôt commencer à coder en java. Mais il nous reste à comprendre **comment s'organise une application**

Java sous Eclipse ?

Observez le schéma suivant :



Notion de package

Nous nous intéressons maintenant au dossier «src».

Eclipse exige la création de sous-dossiers appelés «packages» à l'intérieur de ce dossier «src».

Un package contiendra des classes (des fichiers avec l'extension .java)

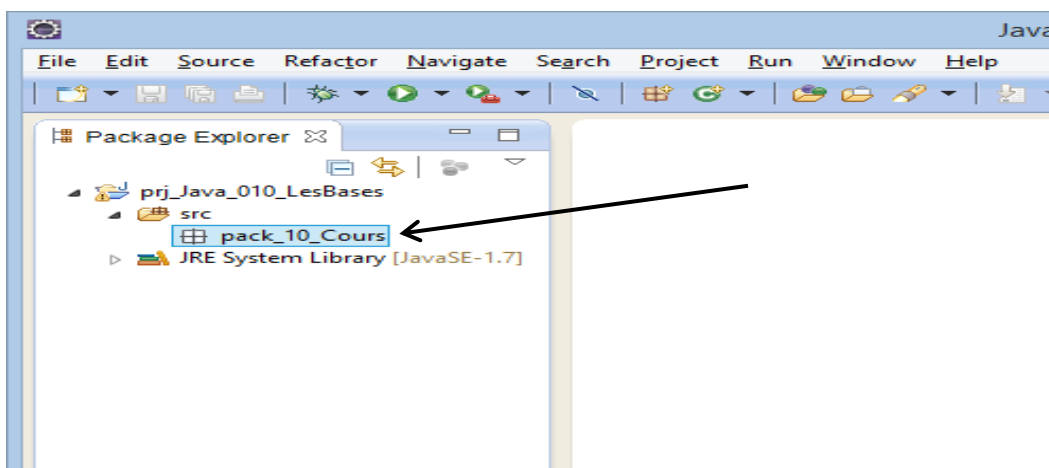
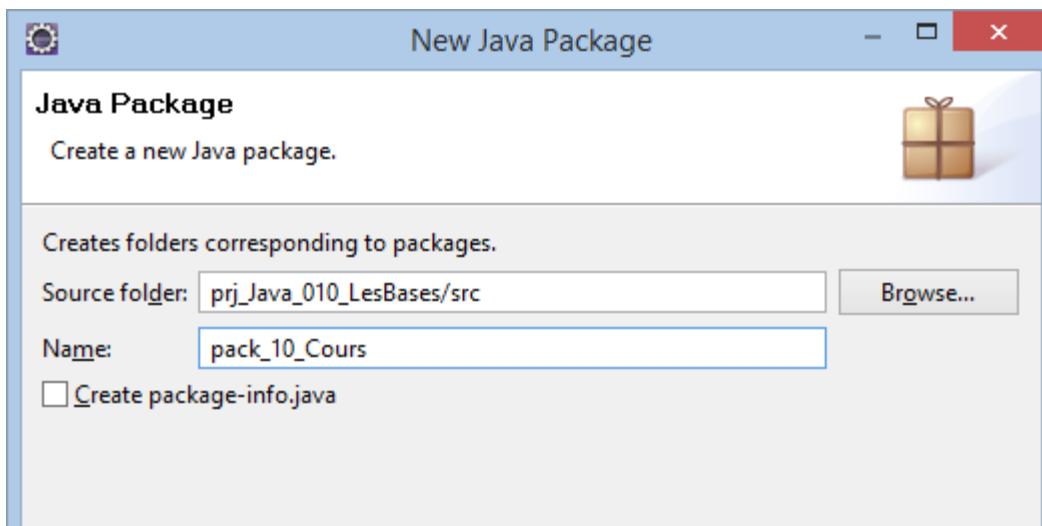
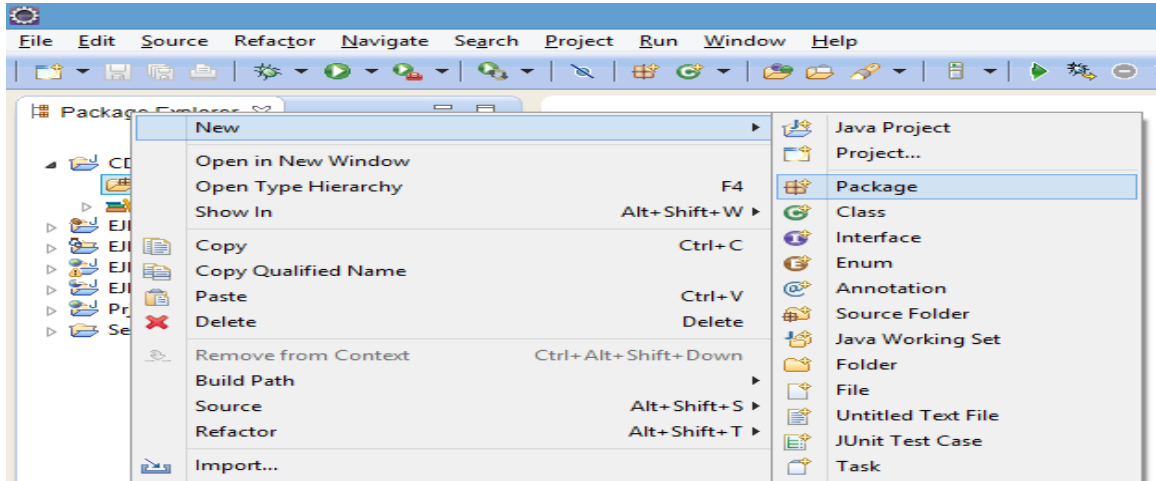
Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

A FAIRE :

Dans le projet que l'on a créé plus haut et nommé «**Java_010_EspaceTravailCours**», cliquez droit sur le dossier «src», puis clic sur «New », puis sur « package » et donner un nom à celui-ci comme ci-dessous :



Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Notion de classe

Dans la prochaine partie de l'apprentissage de java, nous aborderons la notion de classe comme il se doit.

Ici, notre souci premier est d'apprendre à coder en Java : nous allons nous contenter d'une **définition très sommaire**

« **Une classe est un fichier .java qui contient différents morceaux de code java.** Ces morceaux sont appelés « méthodes » et sont l'équivalent des fonctions dans des langages tels que le C, le Visual basic, le PHP, »

Par comparaison :

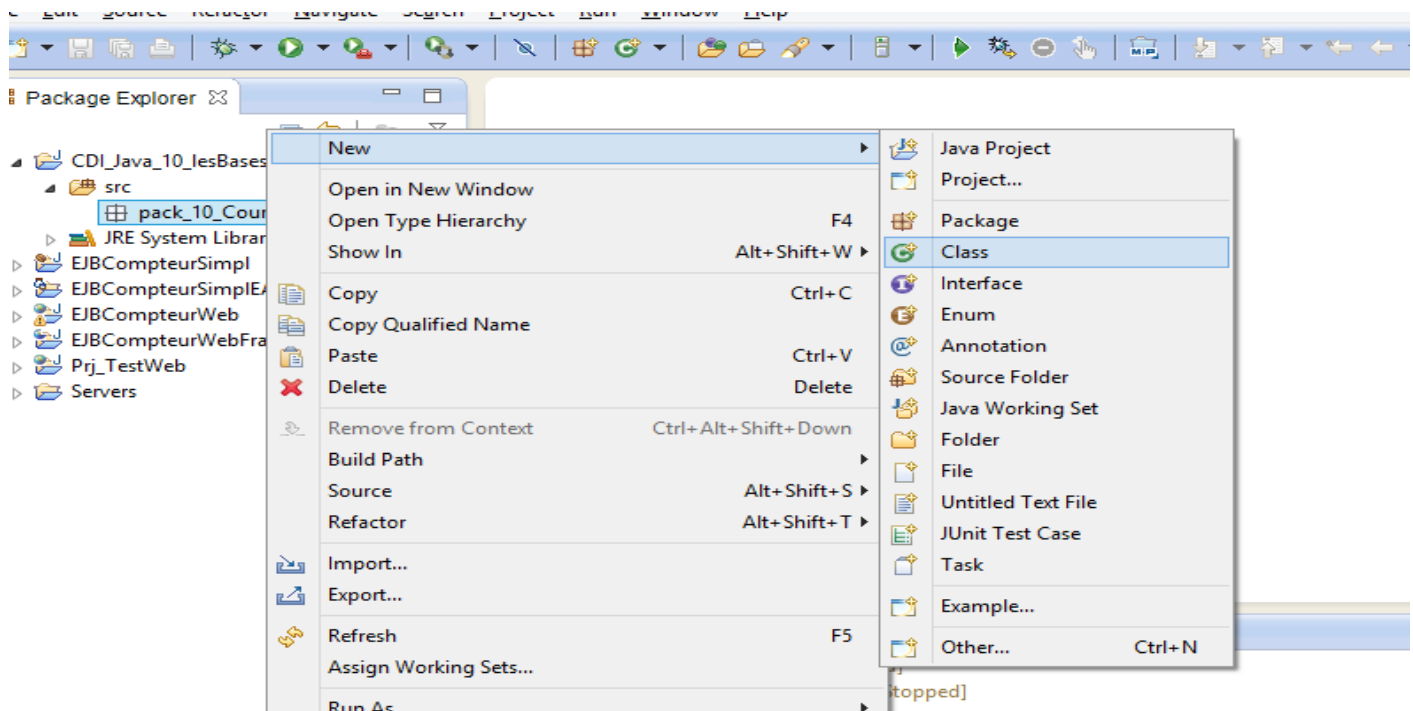
Dans une **application Visual Basic** classique : l'application se décompose en modules et chaque module est un ensemble de fonctions / procédures,

Dans une **application Cobol** : l'application peut être un seul long code ou décomposée en procédures internes appelées souvent sous-programmes internes,

A FAIRE :

Nous allons créer une classe pour tester cette notion :

Cliquez droit sur le package que l'on vient de créer, puis sur « New » et sur « Classe » : donnez un nom à cette classe « **Cours_010_MaPremiereClasse** » :



Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ `public static void main(String[] args)` ←

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

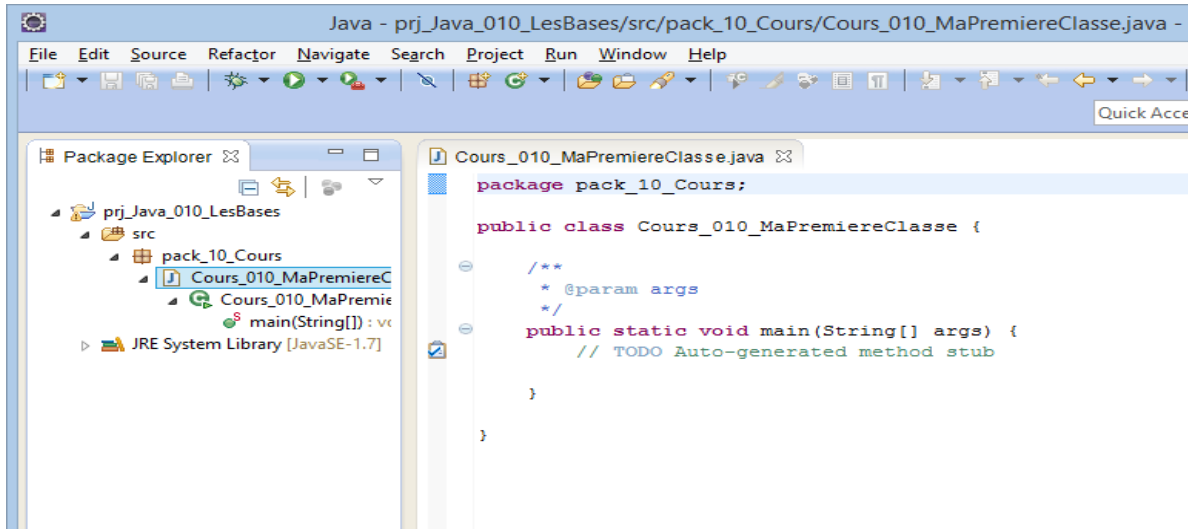
☐ Generate comments

Remarquez que j'ai coché « **public static void main(String[] args)** » avant de cliquer sur «Finish» :

Concepteur Développeur Informatique

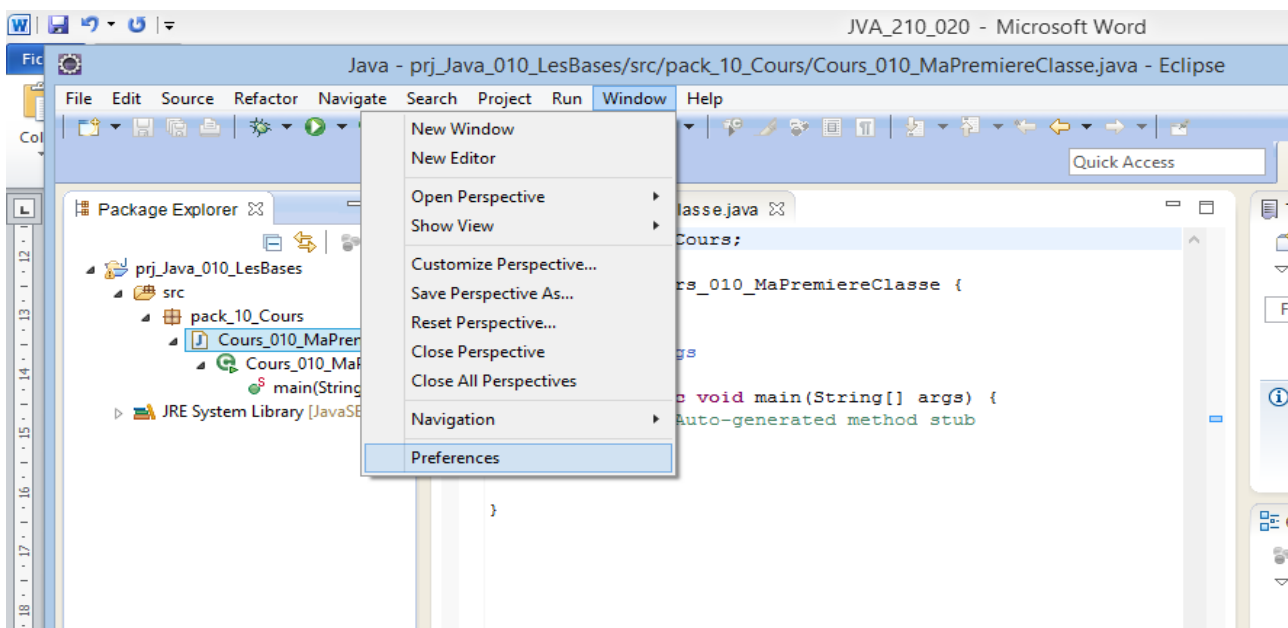
Module : Programmation Java

Bases du Langage



Éclipse m'a généré un code (comme ci-dessus). Ce code n'est pas numéroté. Nous allons **numéroter les lignes** pour bien se comprendre lors des explications sur le code :

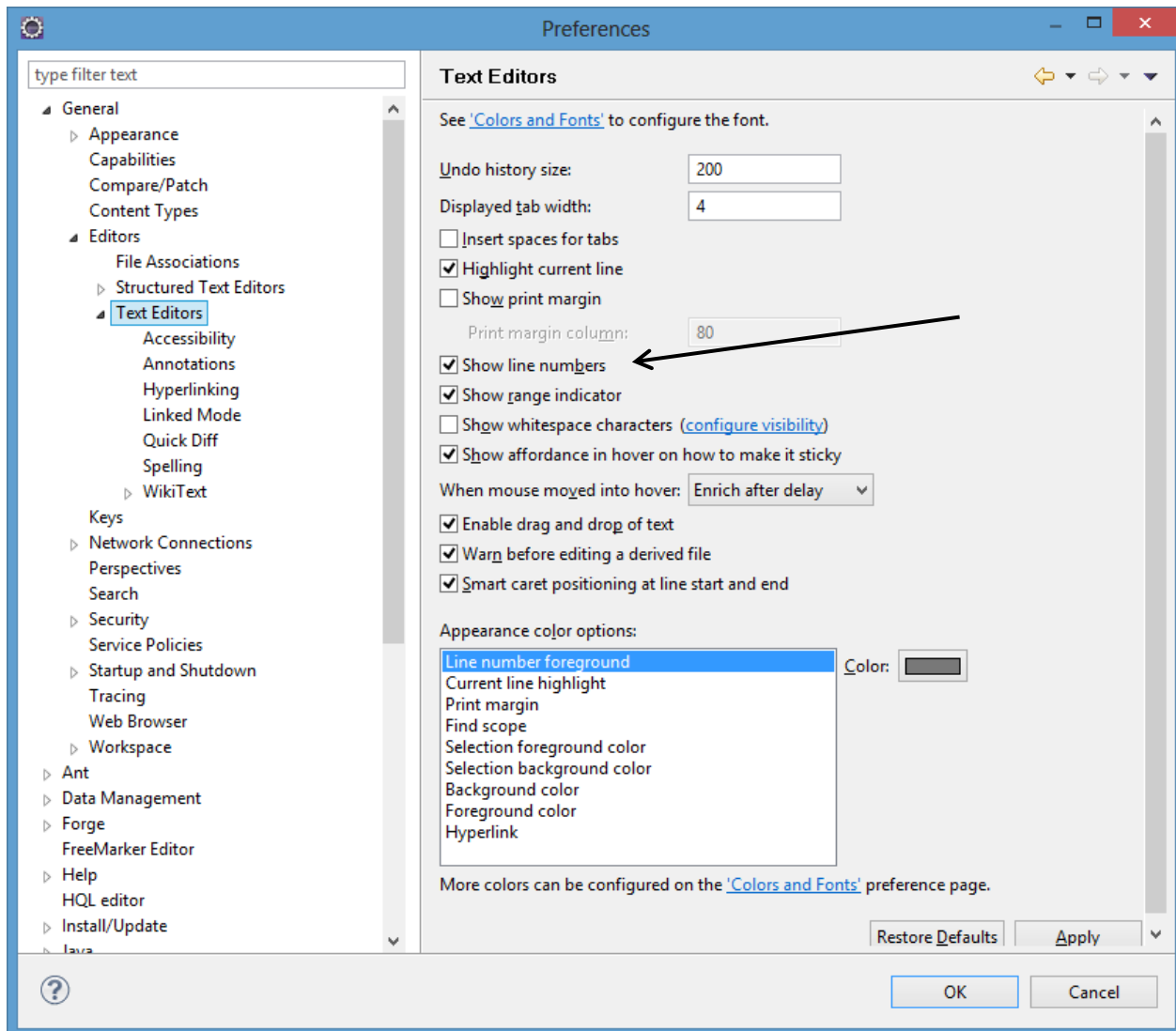
Procédez ainsi :



Concepteur Développeur Informatique

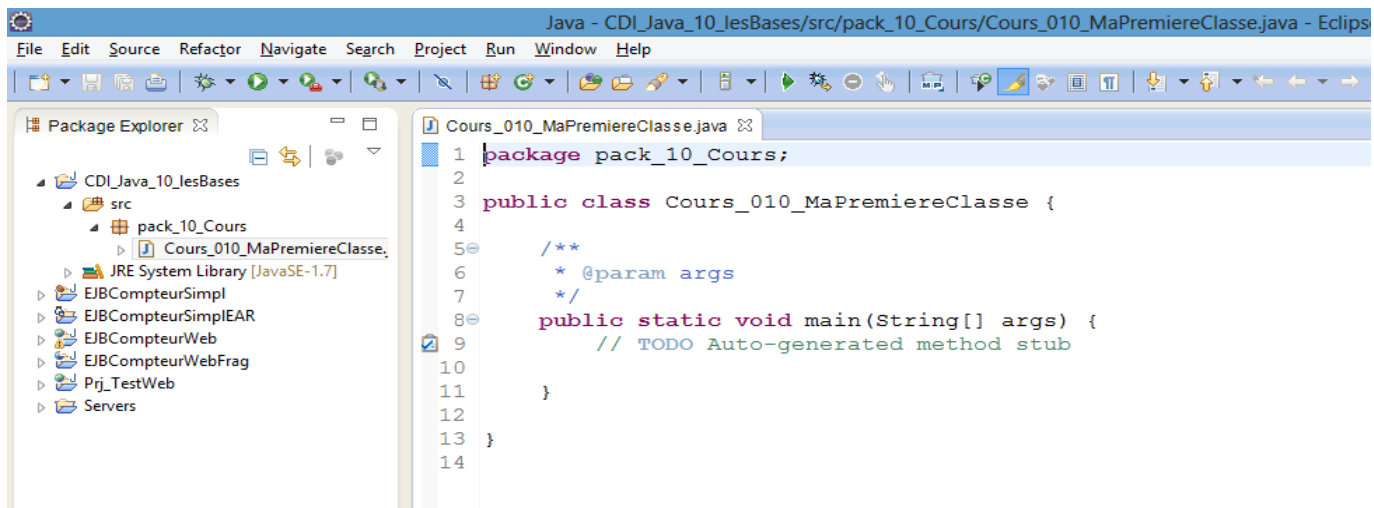
Module : Programmation Java

Bases du Langage



J'ai coché, ci-dessus, «show line numbers».

Cela donne finalement :



Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Nous voyons ci-dessus dans le code d'une classe :

- ➔ Ligne 1 : le nom du package auquel appartient toute classe,
- ➔ Ligne 3 : le nom de la classe
- ➔ Lignes 5 à 7 : lignes commentaires
- ➔ Ligne 8 : nom d'une méthode : nous en parlons ci-dessous.

Notion de Méthode

Une classe java ne peut pas contenir en vrac du code Java. Elle est forcément décomposée en morceaux de codes appelés «méthodes»,

Une méthode :

- ➔ Porte un nom,
- ➔ Elle reçoit des données ou pas : ces données sont appelées « les **paramètres** »,
- ➔ Elle renvoie ou pas une **réponse** lorsqu'elle a fini de s'exécuter,
- ➔ Le nom d'une méthode, dans une classe java, n'est pas unique. Deux méthodes peuvent prendre le même nom : Elles sont différenciées par le nombre et le type de paramètres que reçoit chacune d'elle. (On appellera cela, plus tard : on nommera cette particularité «la surcharge »)

Si on crée, dans une classe, des méthodes (donc des codes séparés), c'est parce que ces codes réalisent des fonctionnalités différentes. Si on crée plusieurs classes : il y aura aussi une raison valable qui explique le découpage.

Exemple de classe et de méthodes :

- ➔ Une classe java pourrait s'appeler « **Facture** » et contenir les méthodes suivantes :
 - Méthode « **créer** » pour créer une facture nouvelle avec ses données : N°, date, n° client,
 - Méthode « **modifier** » pour modifier les données d'une facture existante,
 - Méthode « **calculerTTC** » qui calcule la Montant TTC à partir d'un montant HT et d'une TVA,
 - ;
- ➔ Dans un jeu vidéo, une classe java nommée « Personnage », contiendrait les méthodes suivantes :
 - Méthode «**seDéplacer**» pour faire bouger le personnage dans le jeu,
 - Méthode «**combattre**» pour que le personnage se mette à se battre avec d'autres personnages,
 - Méthode « mourir » qui s'exécute lorsque le personnage aura perdu le combat,
 -

Méthode main

Comme une application Java est un ensemble de codes (méthodes) regroupés dans des classes, une question se pose : Lorsque l'application sera finalisée et que l'on la lance, quel code (quelle méthode) sera exécuté en premier ?

Une décision a été prise : il faut créer une méthode avec un nom imposé « **main** » qui signifie en français la méthode principale. Donc une application commence par exécuter le code de la méthode main.

Projet et application

Lorsque vous êtes en train de fabriquer une application : elle n'est pas terminée, pas totalement testée : c'est un **projet**.

Une **application** est votre projet finalisé avec en plus :

- ➔ Création d'une version exécutable pour certains types de projet,
- ➔ Des configurations à faire : l'application tourne sur le poste du développeur mais il faut qu'elle puisse être

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Accessible aux utilisateurs :

- Configurer les paramètres des serveurs : d'applications, de BD, de
- Configurer les chemins relatifs vers des ressources telles que : des fichiers PDF, des images,
- Répertoire des services Web,
-

Tester le projet

Lorsque vous développez une application, vous avez besoin de tester, au fur et à mesure, les fonctionnalités terminées. Eclipse permet, évidemment, de tester l'application.

Mais selon le type de projet que vous êtes en train de fabriquer, il faudra installer ou pas des composants supplémentaires :

- ➔ Si vous créer un projet Java de type « **console** » : soit il n'affiche rien, soit il affiche sur une console texte (de type fenêtre Dos sous Windows) : rien à installer pour tester,
- ➔ Si vous créer un projet Java de type « **fenêtrée** » : il affiche des données graphiquement dans des fenêtres : rien à installer pour tester,
- ➔ Si vous créer un projet Java de type « **Web** » : il affiche des données graphiquement dans un navigateur Web :
 - Il faut installer un serveur Web + éventuellement un serveur d'applications
- ➔ Si vous créer un projet Java de type « **mobile** » : il affiche des données graphiquement sur des **Smartphones** : dès la création du projet, il faut adjoindre à Eclipse des composants pour réaliser et tester ce type de projet,
- ➔,
- ➔

Déployer l'application

D'abord, aujourd'hui (on est en 2013), une application est rarement réalisée par une seule personne :

En plus des tests réalisés par chaque développeur sur sa partie : il faudra réaliser des **tests d'intégration** pour voir si l'application, dans sa globalité marche.

Exemple :

Une application comme Word 2010 a été réalisée par des dizaines de développeurs Microsoft.

Il a fallu rassembler tous les composants fabriqués par les uns et les autres, les tester pour que cela donne la version finale de Word qu'utilise l'utilisateur tous les jours.

Ensuite, l'application ainsi finalisée sur le (les) poste (s) du (des) développeur(s) n'est pas à mettre aux mains des utilisateurs telle qu'elle. Il y a tout un travail à faire pour la rendre accessible par les utilisateurs finaux :

- ➔ La **stocker** dans l'endroit définitif ou elle sera accessible par les utilisateurs : un endroit du réseau local (LAN), Un lieu d'hébergement Web (type site Internet) ou sur le Cloud,
- ➔ Configurer les **ressources matérielles** (serveurs, équipements réseaux, disques durs,), **logicielles** (serveurs de BD, serveurs d'application, serveurs de messagerie,),
- ➔ Mettre en place une **stratégie de sécurité** et de **confidentialité** (qui accède à l'application ?, qui a le droit de mettre à jour, consulter les données ?,)
- ➔ Mettre en place une stratégie de **sauvegarde** (de la redondance, du mirroring peut être),

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

- ➔ Tester l'application à une échelle proche de la réalité : simuler plusieurs accès à l'application (des dizaines, centaine, milliers, voire millions d'utilisateurs simultanés) : il y a des logiciels qui réalisent ces tests,

Eléments du langage

Nous allons, à partir de maintenant, commencer à coder en Java et à tester notre code.

Pour cela, nous allons nous organiser ainsi :

Nous allons créer à chaque apprentissage, une classe Java nommée « Cours_0XXX_aaaaaaaaaaaa »

Où « 0xxx » sera un nombre progressif et « aaaaaaaa » un texte qui suggère ce que contient la classe,

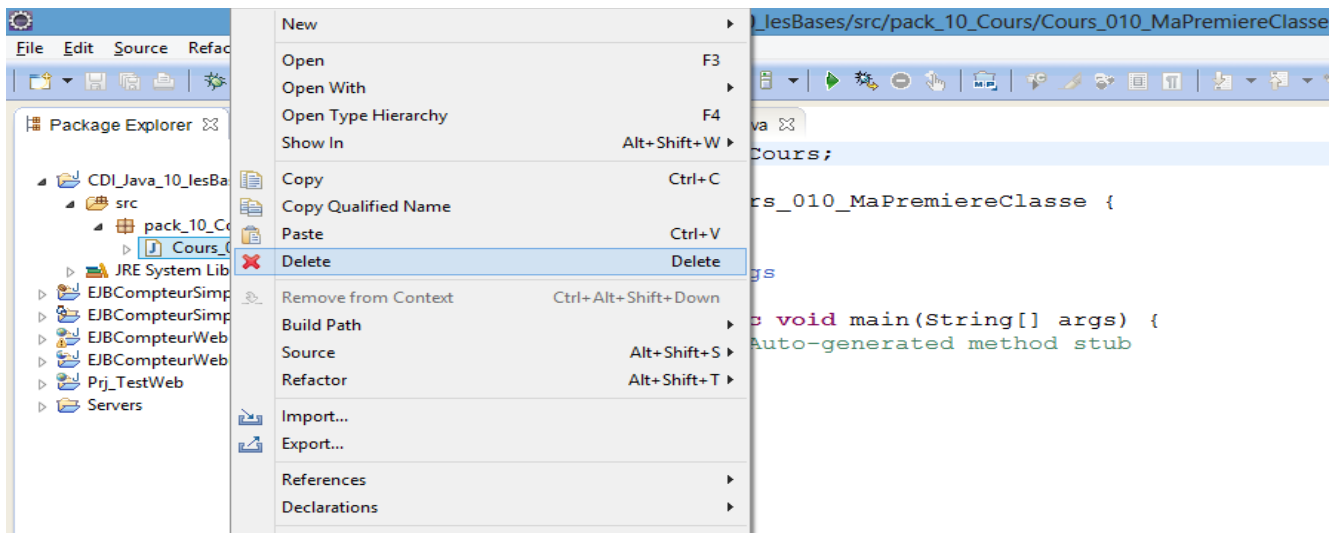
Chaque classe contiendra la méthode « main » pour pouvoir tester le code :

Cela signifie « **une classe qui ne contient pas de méthode main ne se teste pas. Elle est uniquement utilisée par les autres classes** »

L'affichage se fera dans une vue d'Eclipse appelée « console » :

Premier code en Java

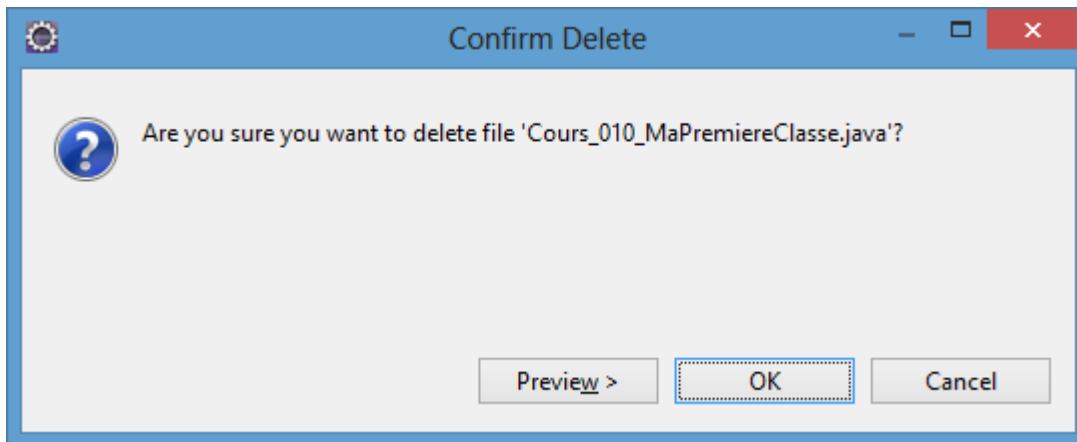
- ➔ Supprimez la classe créée précédemment :



Concepteur Développeur Informatique

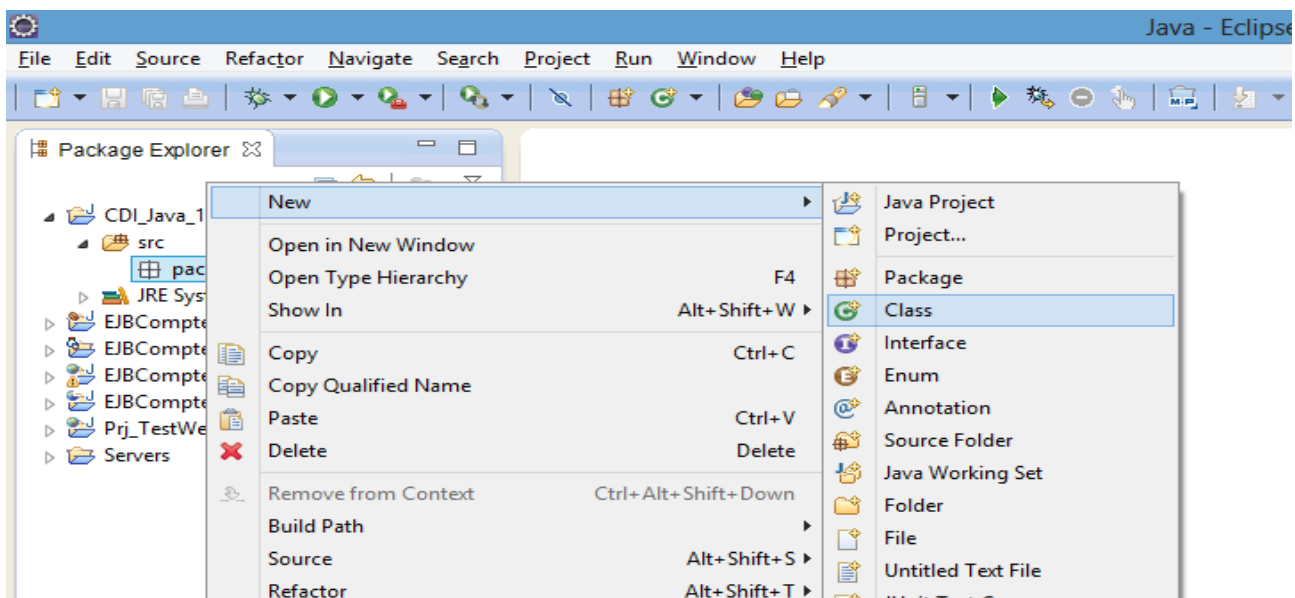
Module : Programmation Java

Bases du Langage



On confirme la suppression.

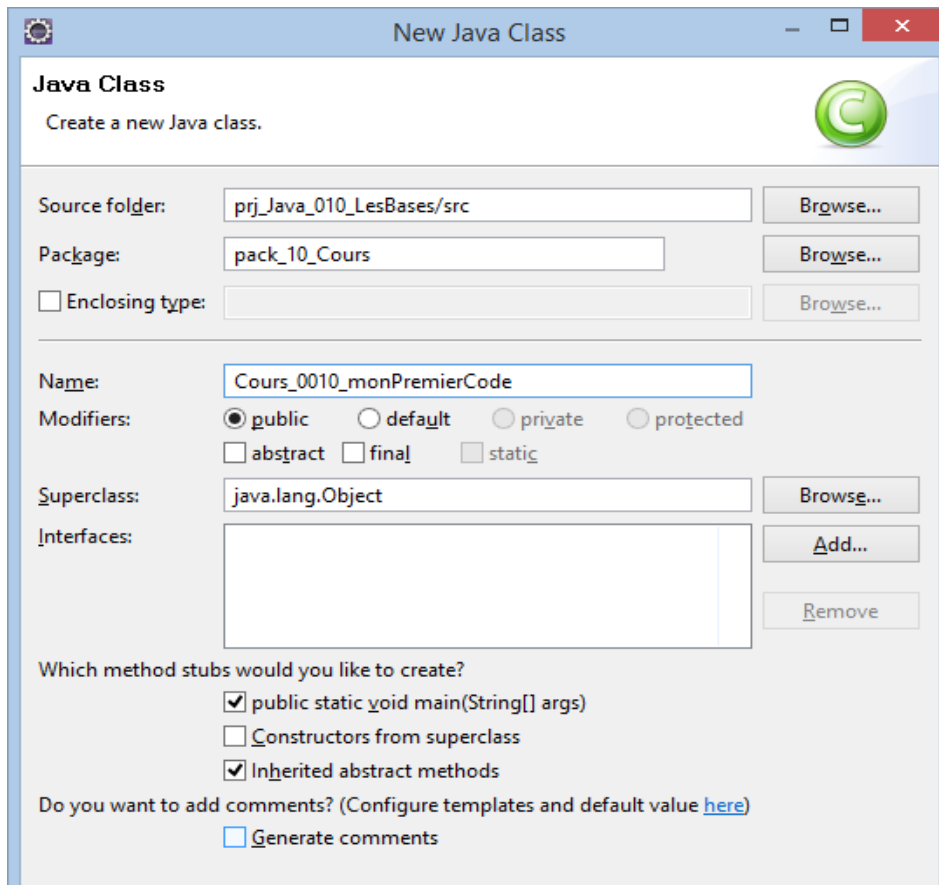
➔ Créons maintenant notre premier code qui consiste à afficher un texte sur la console :



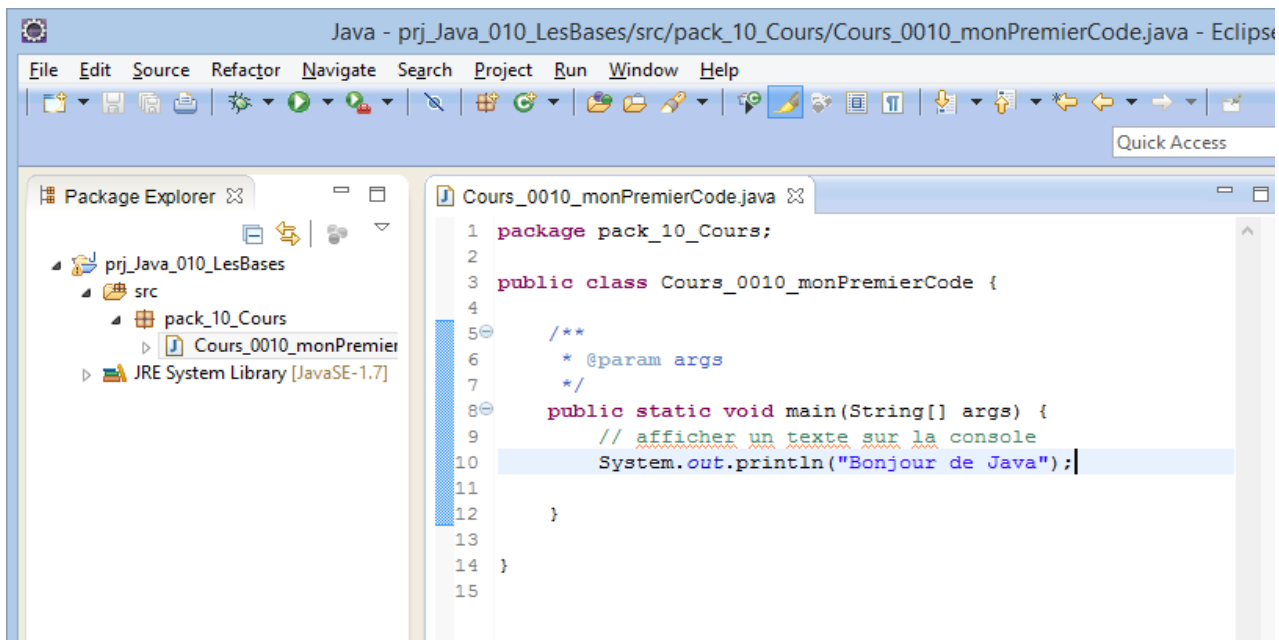
Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Modifiez le code généré pour obtenir ceci :

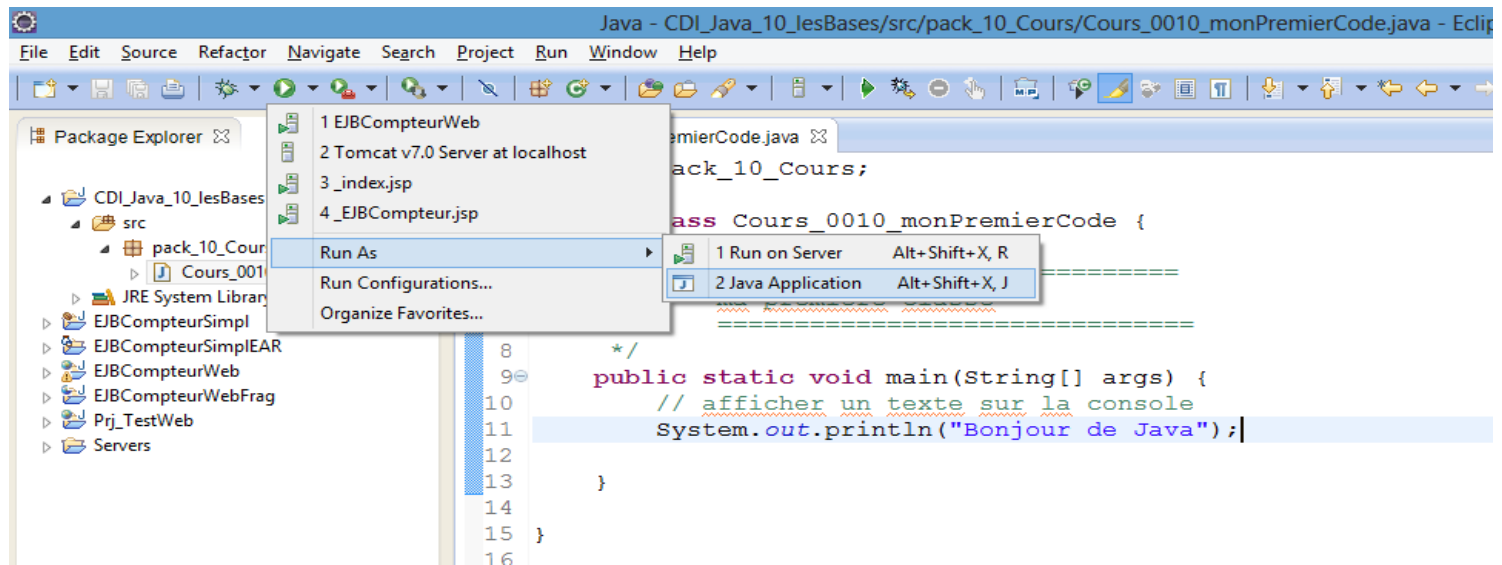


Concepteur Développeur Informatique

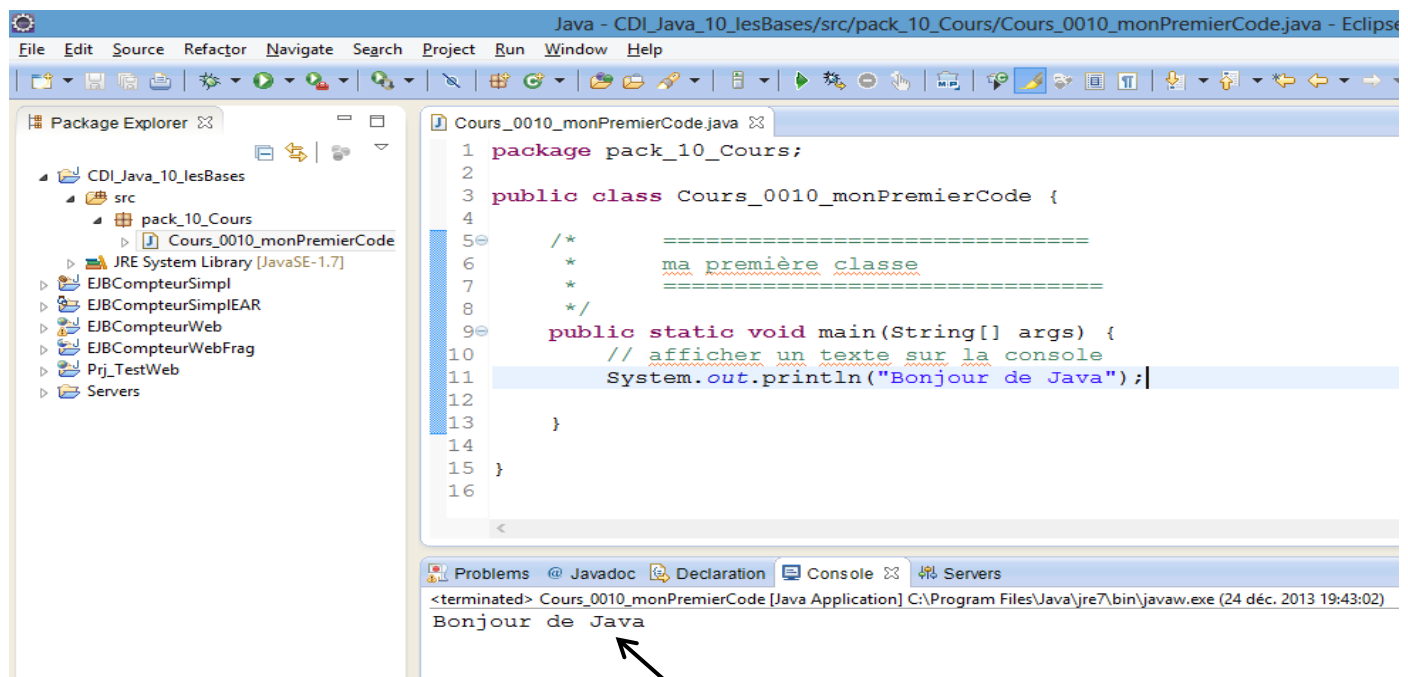
Module : Programmation Java

Bases du Langage

Testez ce code de la façon suivante :



Cela donne sur une vue, en bas, appelée «Console» :



Attention :

- ➔ Pour le langage Java, la casse est importante : **System** n'est pas la même chose que «**system**»
- ➔ Ne vous posez pas des questions sur la ligne 9 : «**public static void**» puis «**(String [] args)**» sont des termes qui peuvent apparaître obscures et inintelligibles pour le débutant : ne cherchez pas à les comprendre pour le moment. Cela se fera plus tard.

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Commentaires

Un commentaire est du texte pour documenter le code java et il n'est donc pas exécuté comme instructions.

Les commentaires sont très utiles :

- Pour expliquer ce que fait le code,
- Surtout quand le code est volumineux,
- Quand le code est réexaminé plus tard,
- Quand le code est partagé et/ou modifié par plusieurs développeurs,
-

Un code est utile quand il explique la fonctionnalité codée et pas le code lui-même : par exemple

Un commentaire qui dit « boucle sur les factures » explique juste qu'il y a une boucle alors qu'un commentaire qui dit « impression de toutes les factures du jour » est un commentaire fonctionnel,

Commentaire sur une seule ligne

On utilise // :

```
// Calcul des heures supplémentaires
```

Commentaire sur plusieurs lignes

On utilise /* au début de la première ligne et */ à la fin de la dernière ligne :

```
/*
 *   ci-dessus : nous examinons les commandes qui
 *   peuvent être satisfaites par le stock actuel
 *   celles qui ne le peuvent pas génèrent une alerte
 */
```

Les étoiles dans les lignes intermédiaires sont générées par Éclipse et ne sont pas obligatoires.

Syntaxe d'une instruction élémentaire en Java

Une instruction élémentaire en java est une instruction qui finit par un point-virgule.

Par exemple :

```
x=y ;
```

Ou

```
char caractere ;
```

Une instruction bloc est un ensemble de code java encadré par les caractères { et }

Par exemple :

```
if(nbreCommandes==0)
{
    // .....
    // .....
}
```

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Ou

```
for (int compteur=0 ; compteur < nbreJoueurs; compteur++)  
{  
    /*  
    *  
    *  
    *  
    *  
    */  
}
```

La première ligne et la dernière ligne d'un bloc ne finissent pas par un « ; »

Les instructions à l'intérieur du bloc finissent par un « ; »

Une instruction hors-piste

Une instruction java doit, absolument, être codée à l'intérieur d'une méthode. Sinon elle est refusée.

Afficher des données sur la console

Un code doit souvent afficher des données à l'utilisateur.

Selon le type de projet Java créé, l'affichage peut être sous forme textuelle ou graphique.

Ici, nous nous intéressons à l'affichage textuel.

L'instruction « `System.out.println("Bonjour de Java");` » permet d'afficher un texte fixe (comme ci-dessus) ou des données variables (à voir plus loin).

Erreurs de saisie

Lorsque vous faites une erreur de saisie (par exemple : il manque le ; ou l'instruction est mal orthographiée,),

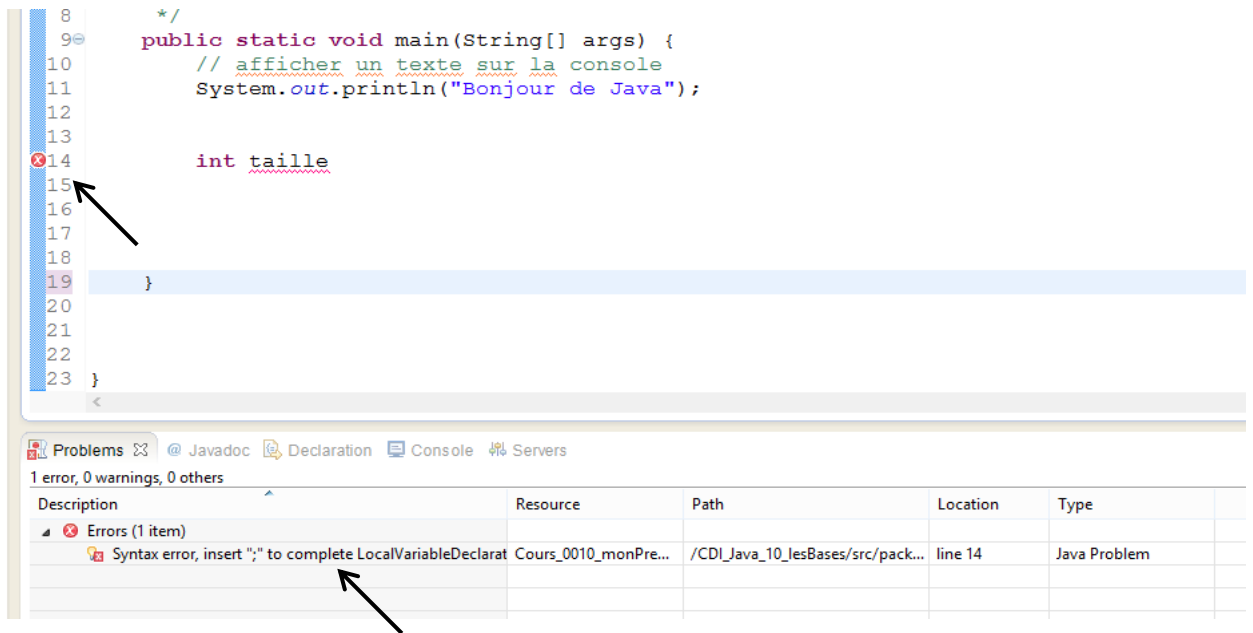
Éclipse vous signale l'erreur avec une croix rouge à gauche de l'instruction et vous affiche un texte descriptif de l'erreur dans une vue «Erreur».

Exemple 1 : il manque ; à la fin de l'instruction

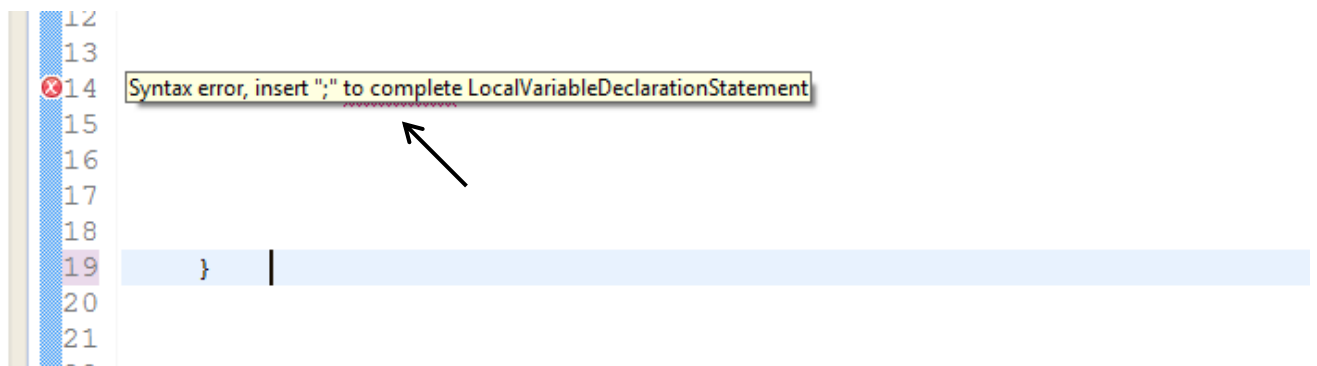
Concepteur Développeur Informatique

Module : Programmation Java

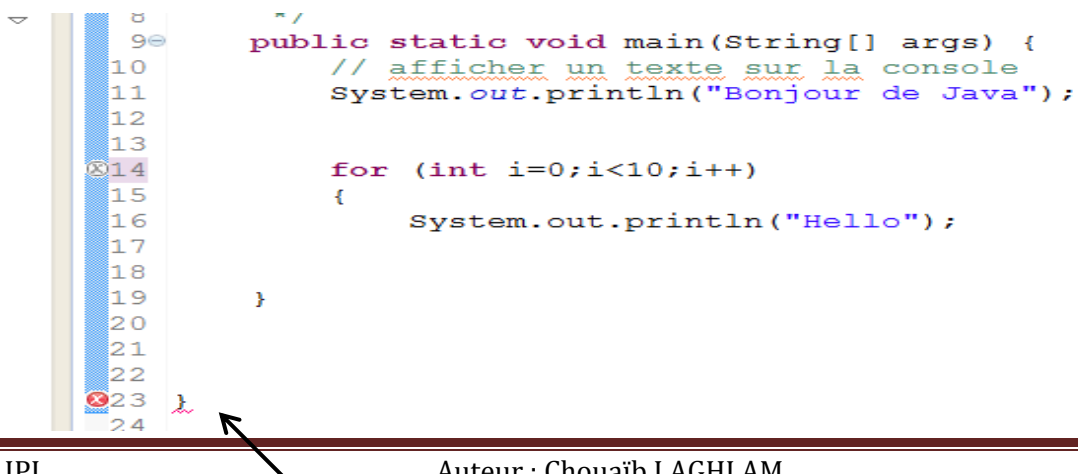
Bases du Langage



Si vous approchez la souris de la croix rouge, qui se trouve à gauche de l'instruction erronée, Eclipse vous affiche une info-bulle qui explique l'erreur :



Exemple 2 : il manque une } à la boucle for



Concepteur Développeur Informatique

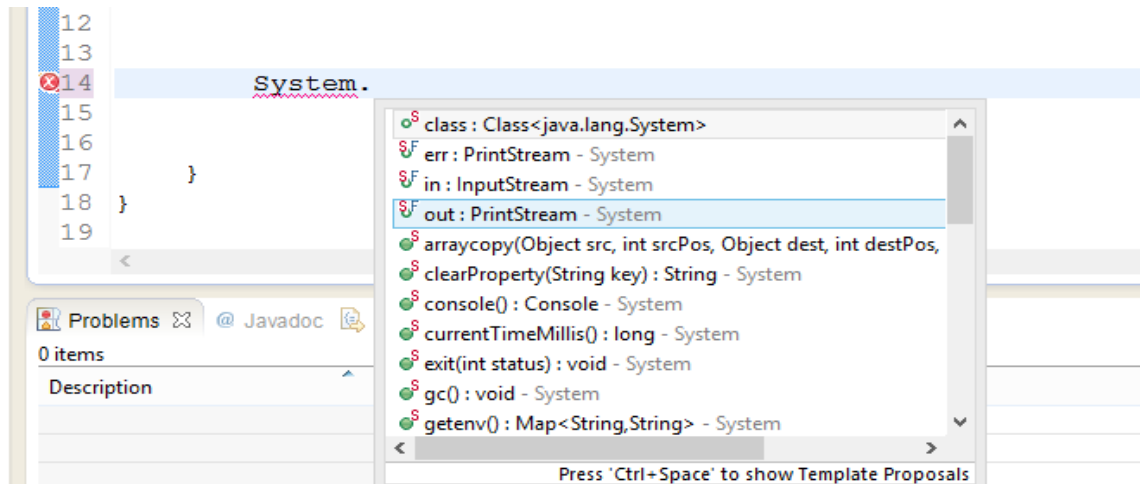
Module : Programmation Java

Bases du Langage

Aide à la saisie d'instructions

Éclipse aide le développeur, lors de la saisie d'instructions, par un pré affichage intuitif :

Exemple : je souhaite taper l'instruction, «**System.out.println(...)** »,
Je commence à taper : System. Et Eclipse m'affiche la suite possible



Java aime la casse

Le langage java fait la différence entre les minuscules et les majuscules.

Par exemple «**System.out.println(...)**» sera accepté

Mais «**system.out.println(...)**» sera refusé car le s est en minuscule.

Variables de type primitif

Comme dans tout langage informatique, nous avons besoin de **stocker** des données **momentanément** en mémoire.

Exemples :

- Lors d'une connexion d'un client d'un site Web, conserver son identifiant et son mot de passe pendant un laps de temps,
- Lors d'un achat sur internet, stocker les articles mis dans le panier en attendant la validation ou pas de l'achat,
- Stocker en mémoire un SMS pendant que l'utilisateur le saisit,
-

Nous parlons ici de **stocker momentanément** : cela signifie que la donnée est gardée en mémoire de la machine Au maximum pendant la durée d'exécution du programme qui utilise cette donnée,

Si nous souhaitons **stocker** pour plus **longtemps** une donnée : il y a plusieurs moyens de le faire selon le contexte :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

- Stocker dans un fichier sur le disque dur,
- Stocker dans une base de données,
- Stocker dans des cookies,
- Stocker sur le Cloud,
- Etc.

Variable

Le stockage momentanée d'une donnée dans un endroit de la mémoire de la machine est appelée «variable»,

Le développeur **déclare une variable** : cela revient à réserver une place en mémoire pour stocker une donnée

Java exige, pour déclarer une variable, de :

- Donner un **nom** à la **variable**,
- De donner un **type** à la **variable**,
- De donner, parfois, la **longueur** en octets à donner à la **variable**.

Nom de variable

Une variable doit avoir un nom unique dans le contexte utilisé (nous parlerons du contexte bientôt),

Le nom doit commencer par une lettre de l'alphabet,

Un nom ne peut contenir un espace ou un caractère spécial (le tiret du 8 est accepté),

Les accents sont à prohiber car on travaille de plus en plus en équipes internationales : les accents ne sont pas acceptés dans toutes les langues ;

Préférez «terminee » à «~~terminée~~»

Type de variable

Java a besoin, à l'avance, de savoir quel type de données, vous allez stocker en mémoire.

Pourquoi ?

Car Java va calculer la place que votre donnée va prendre en mémoire.

Types primitifs

Java a répertorié certains types de données les plus utilisés et sait à l'avance quelle longueur il leur donnera. On appelle ce type de données : les **types primitifs**.

Voici un tableau qui récapitule les huit types primitifs :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	'\u0000' ? '\uffff' (0 à 65535)
byte	Entier très court	1	-128 ? +127
short	Entier court	2	-32 768 ? +32 767
int	Entier	4	-2^{31} ? $-2,147 \times 10^9$? $+2^{31}-1$? $2,147 \times 10^9$
long	Entier long	8	-2^{63} ? $-9,223 \times 10^{18}$? $+2^{63}-1$? $9,223 \times 10^{18}$
float	Nombre réel simple	4	$\pm 2^{-149}$? $1,4 \times 10^{-45}$? $\pm 2^{128}$? $3,4 \times 10^{38}$
double	Nombre réel double	8	$\pm 2^{-1074}$? $4,9 \times 10^{-324}$? $\pm 2^{1024}$? $2,971 \times 1,8 \times 10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Exemples :

But	Exemple
Stocker le nombre d'étudiants dans une classe	short nbreEtudiants ;
Stocker le nombre d'habitants d'un pays	int population ;
Mémoriser la situation familiale par un seul caractère : D pour divorcé(e), M pour marié (e),	char situationFamiliale ;
Garder en mémoire le montant d'une facture	float mntFacture ;
Se rappeler si le solde est créditeur ou pas	boolean estCrediteur ;
.....	

Le type String

Ce type permet de stocker une chaîne de caractère de longueur variable.

Par exemple :

Pour stocker le nom d'une personne, une adresse postale, une adresse email,

ATTENTION : le type char permet de stocker un seul caractère et pas des caractères,

Les autres types de données, que l'on déclarera plus tard, ne sont pas considérés comme primitifs : leur déclaration sera vue le moment venu.

Déclaration d'une variable

Syntaxe : type de la variable nom de la variable ;

Exemples

int i ;

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
long nbre_molecules ;  
boolean enregistrementEstTermine ;
```

Initialisation d'une variable

Si au moment de la déclaration, je connais la valeur initiale que je veux donner à la variable, je déclare et j'initialise ainsi :

```
int compteur=0 ;
```

Cette instruction est équivalente à :

```
Int compteur;  
compteur=0 ;
```

Affectation d'une variable

Mettre une valeur dans une variable s'appelle «affecter une variable».

La valeur à mettre dans une variable peut être une constante ou le contenu d'une autre variable.

Exemples :

Mettre dupont dans une variable nommée nom et de type String	String nom; nom="dupont"; Remarquez les guillemets (ou double quotes)
Mettre le caractère % dans une variable nommée x et de type char	char x; x='%'; Remarquez les simples quotes
Stocker dans la variable montant le nombre 17,25	float <u>montant</u> ; montant=17.25f; remarquez le . et le f
Se rappeler qu'une personne est célibataire	boolean estCelibataire; estCelibataire=true;
Mettre le contenu de la variable A dans la variable B	String A="Alain"; String B="Bernard"; B=A; A et B contiendront Alain

ATTENTION : **une variable non affectée (ne contient aucune valeur) ne peut pas être utilisée**
: affecter une autre variable, ne peut être testée.

Portée d'une variable

Une variable existe lorsque l'on peut copier, modifier son contenu. On dit qu'elle est accessible.

Une variable disparaît d'une façon certaine lorsque le programme s'arrête de s'exécuter.

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Elle sera recréée de nouveau, lors de la prochaine exécution du programme, avec les mêmes ou d'autres valeurs,

Mais souvent, elle n'est **accessible que dans un contexte donnée** :

On parle alors de la **portée d'une variable** :

- Dans quel portion de code est-elle accessible ?
- Quand est-elle supprimée ?

Voici **quelques exemples** :

Exemple 1 :

```
1 package pack_10_Cours;
2
3 public class Cours_0010_monPremierCode {
4
5     /*      =====
6      *      ma première classe
7      *      =====
8      */
9     public static void main(String[] args) {
10         int x=1;
11         if (x>0)
12         {
13             int y=12;
14         }
15         y=0;
16         x=17;
17
18     }
19 }
20
```

Tableau récapitulatif des déclarations des variables ci-dessus :

Déclaration variable	Contexte de déclaration	Contexte d'utilisation	Problème
10 int x=1;	x est déclaré dans la méthode main	x sera utilisé à l'intérieur de cette méthode main	
13 int y=12 ;	y est déclarée dans un contexte plus restreint que celui de la méthode main : y est déclarée dans un bloc de test if	Y peut être utilisé à l'intérieur du bloc if () { }	
15 y=0 ;		y utilisée en dehors du bloc du if	y ici n'est pas reconnu en dehors du bloc if
16 x=17 ;		x est utilisé et accepté car x est utilisé à l'intérieur de la méthode main	

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Exemple 2 :

```
30     int var=1;
31     String ville="Paris";
32     if(var>1)
33     {
34         String ville="Marseille";           // déjà déclarée en global
35     }
36     System.out.println("Ville : "+ville);    // reconnaît ville
37     for (int i=0;i<13;i++)
38     {
39         String ville="Lille";               // déjà déclarée en global
40     }
41 }
42 }
```

Ci-dessus, nous avons une classe qui contient une méthode « **main** », qui elle-même, contient un **bloc if** un **bloc for**

Déclaration variable	Contexte de déclaration	Contexte d'utilisation	Problème
31 String ville="Paris";	ville est déclarée dans la méthode main	ville sera utilisée à l'intérieur de cette méthode main et dans les blocs inclus dans cette méthode	
34 String ville="Marseille";	Variable de même nom déjà déclarée dans un contexte supérieur : tentative de la déclarer dans un contexte bloc if inclus dans le contexte de la méthode main	La variable peut être utilisée ainsi : Ville="Marseille"; Mais pas déclarée : String ville="Marseille";	Java dit qu'il y a duplication de variables !!
39 String ville="Lille";	Même problème que la ligne 34		

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Exemple 3 :

```
43 |
44 | /* =====
45 | *      EXEMPLE 3
46 | *      =====
47 | */
48 | int n=10;
49 | if(n>19)
50 | {
51 |     String pays="France";
52 |     System.out.println("Pays : "+pays);    // pourquoi pays ok ?
53 | }
54 | System.out.println("Pays : "+pays);        // pourquoi pays pas ok ?
55 | for (int k=0;k<28;k++)
56 | {
57 |     String pays="Argentine";              // pourquoi pays ok ?
58 | }
59 | System.out.println("Pays : "+pays);        // pourquoi pays pas ok ?
60 | }
61 | }
```

Pourquoi, ci-dessus, les lignes 54 et 59 sont en erreurs !! !!

Déclarer une constante

Si nous avons besoin de stocker une valeur constante en mémoire pour l'utiliser dans le code mais sans jamais changer son contenu, alors il faut faire une déclaration différente d'une variable.

- On déclare, dans une méthode, une constante avec le mot «**final**» pour l'utiliser uniquement dans cette méthode,
- Si on souhaite utiliser une constante dans toutes les méthodes d'une classe, on la déclare avant la 1^{ère} méthode et avec «**static final**»

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
10      *      Modifié le :
11      *      Modifié par :
12      *      =====
13      */
14      /* une déclaration globale de constante : valable pour toutes les
15         méthodes de cette classe
16      */
17      static final double pi=3.1415;
18      // méthode main
19      public static void main(String[] args) {
20          // fixer la constante pi dans cette méthode
21          final double pi=3.14;
22          int rayon=12;          // variable de type entier
23          System.out.println("valaue de pi : "+pi); // quelle valeur de pi s'affiche
24          // je tente de modifier une constante
25          pi=4.0;
26          // ....
27          double aire= rayon * rayon * pi;          // calcul de surface d'un cercle
28
29      }
```

Problems @ Javadoc Declaration Console Servers
<terminated> Cours_0060_declarationsConstantes [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 08:48:51)
valaue de pi : 3.14

- o Ligne 19 : on déclare une constante globale pi,
- o Ligne 21 : on déclare une constante nommée pi dans une méthode,
- o Ligne 22 : on déclare une variable nommée rayon,
- o Ligne 23 : quelle valeur de pi va s'afficher ?
- o Ligne 25 : java n'accepte pas que l'on tente de modifier une constante : Donc Erreur.

Une instruction java est une fonction qui vient de quelque part

Une instruction java, à ce stade, peut-être :

- o Une déclaration d'une variable : par exemple **String adresse ;**
- o Un bloc de test ou de boucle : par exemple **if(.....) {}**
for(.....) {}
- o L'exécution d'un code fabriqué par les inventeurs de Java : appel d'une méthode :
Par exemple **System.out.println(.....)**

Dans le dernier cas cité, on appelle bien une méthode (ou une fonction) qui s'appelle **println()**

Il existe des centaines, des milliers de méthodes en Java. Chacune sert à quelque chose de précis.

C'est pour cela, que certains disent, que java est compliqué par référence à des langages dont le nombre D'instructions est fini.

Or il n'en est rien : un développeur Java apprend les instructions de base du langage et au fur et à mesure de ces besoins : il découvre les méthodes nécessaires

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Pour organiser ces milliers de méthodes et les rendre facilement trouvable : il a été décidé de les rassembler en branches partant d'une racine commune.

On passe d'une branche à une autre par le caractère point.

Par exemple : **System.out.println("") ;**

On exécute la méthode « println » en passant par les branches System et out.

Par exemple : **java.io.BufferedReader ent = ;**

Demander des données à l'utilisateur

En Java, on a bien prévu les méthodes « println » et « print » pour afficher des données sur la console,

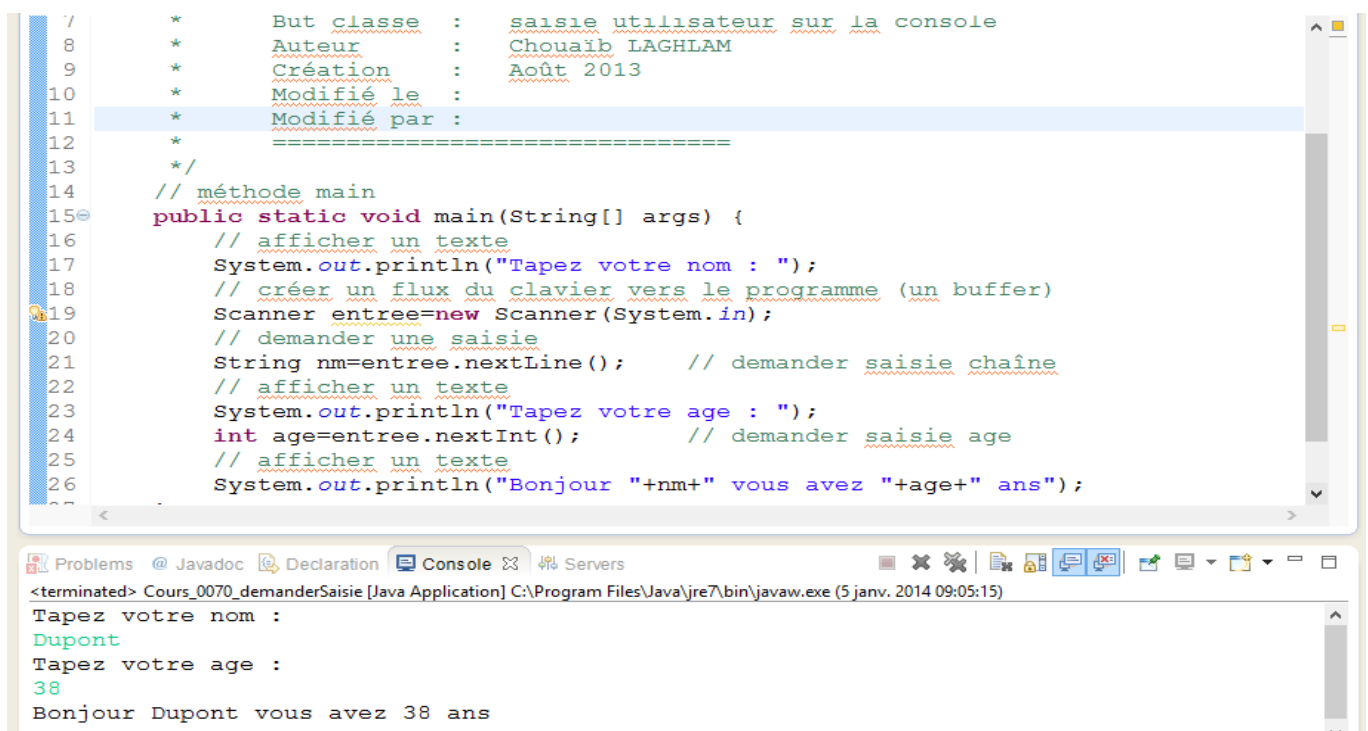
Mais pour la saisie des données (demander des informations à l'utilisateur), il y a eu plusieurs solutions au fil du temps,

L'une des premières solutions proposées est la classe **Scanner**.

Classe scanner

A ce stade du cours, nous ne savons pas ce qui est une classe, mais nous allons utiliser une syntaxe pour demander à l'utilisateur de nous fournir une information.

Exemple : demander à l'utilisateur son nom, son âge et lui dire bonjour



```
1  *      But classe :   saisie utilisateur sur la console
2  *      Auteur      :   Chouaib LAGHLAM
3  *      Création    :   Août 2013
4  *      Modifié le   :
5  *      Modifié par :
6  *      =====
7  */
8
9  // méthode main
10 public static void main(String[] args) {
11     // afficher un texte
12     System.out.println("Tapez votre nom : ");
13     // créer un flux du clavier vers le programme (un buffer)
14     Scanner entree=new Scanner(System.in);
15     // demander une saisie
16     String nm=entree.nextLine();    // demander saisie chaîne
17     // afficher un texte
18     System.out.println("Tapez votre age : ");
19     int age=entree.nextInt();       // demander saisie age
20     // afficher un texte
21     System.out.println("Bonjour "+nm+" vous avez "+age+" ans");
22 }
```

Problems | Javadoc | Declaration | Console | Servers

<terminated> Cours_0070_demanderSaisie [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 09:05:15)

Tapez votre nom :
Dupont
Tapez votre age :
38
Bonjour Dupont vous avez 38 ans

Explications du code ci-dessus :

- Lors de la saisie de la **ligne 19**, Eclipse met en rouge le mot « Scanner » car il ne sait pas d'où vient la définition de cette classe : cliquez sur la croix rouge à gauche : Eclipse va vous suggérer d'importer cette

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

classe du chemin « java.util.Scanner » : faites un double-clic dessus : cela va générer une ligne commençant par « import » tout en haut.

- Ligne 17 :
 - On affiche à l'utilisateur un texte pour qu'il sache ce qu'il va saisir,
- Ligne 19 :
 - On prépare la demande de saisie : on fabrique un objet nommé entrée : cela permet de recevoir la frappe au clavier,
- Ligne 21 :
 - On demande (grâce à la méthode nextLine) la saisie d'un texte et on décide de le mémoriser dans la variable nm,
- Ligne 24 :
 - On demande (grâce à la méthode nextInt) la saisie d'un entier et on décide de le mémoriser dans la variable age,
- Ligne 26 :
 - On dit bonjour en réaffichant les saisies,
 - On mélange dans l'instruction du texte fixe « Bonjour : » et le contenu d'une variable
On utilise l'opérateur + qui ne signifie pas ici une addition mais une concaténation, c'est-à-dire un Assemblage de chaînes de caractères.

Réaliser des tests

Comme les autres langages, Java possède plusieurs instructions pour réaliser des tests,

Un test est :

- Comparer le contenu d'une ou de plusieurs variables à des valeurs,
- Le constat du test est toujours « vrai » ou « faux » : en anglais «**true**» ou «**false**»,
- Décider des instructions à exécuter selon le constat de la comparaison :
 - Quelles instructions à exécuter si le constat est «true»,
 - Quelles instructions à exécuter si le constat est «false»,

Nous nous contentons ici d'utiliser les deux instructions les plus utilisées pour les tests : le **if** et le **switch**.

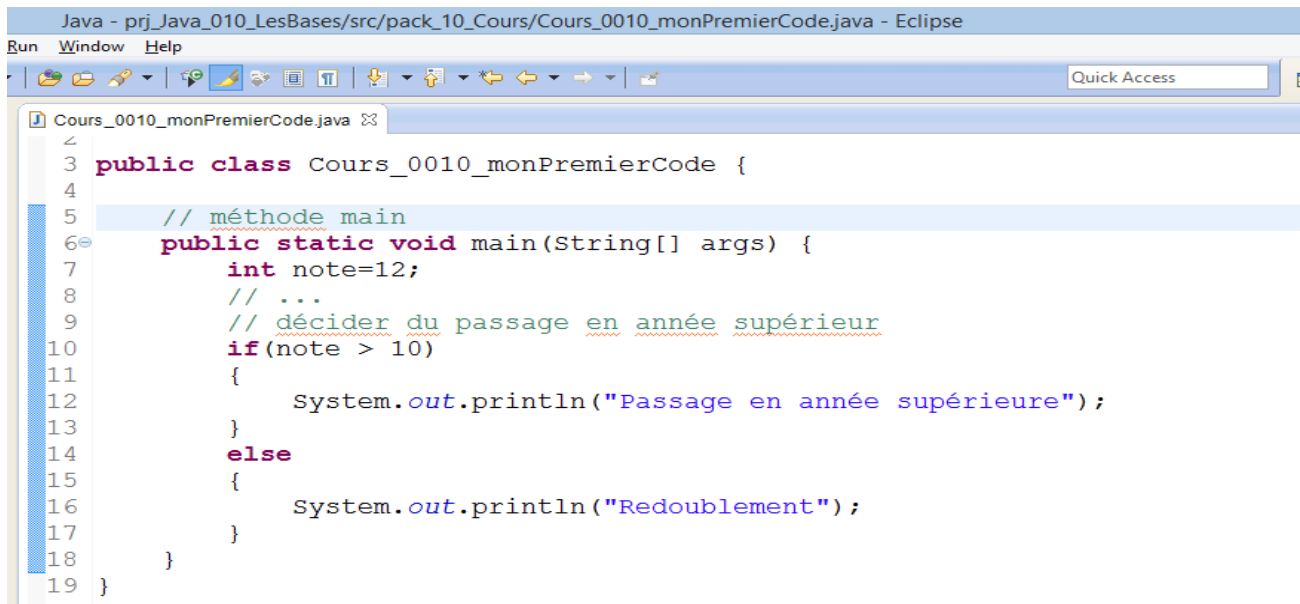
If

Exemple 1 : comparer une variable à une valeur fixe

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



```
Java - prj_Java_010_LesBases/src/pack_10_Cours/Cours_0010_monPremierCode.java - Eclipse
Run Window Help
Cours_0010_monPremierCode.java
2
3 public class Cours_0010_monPremierCode {
4
5     // méthode main
6     public static void main(String[] args) {
7         int note=12;
8         // ...
9         // décider du passage en année supérieur
10        if(note > 10)
11        {
12            System.out.println("Passage en année supérieure");
13        }
14        else
15        {
16            System.out.println("Redoublement");
17        }
18    }
19 }
```

Explications sur le code ci-dessus :

- Ligne 19 :
 - La variable « note » est supposée déclarée et affectée par une variable,
 - On se demande si le contenu de la variable « note » est supérieur à 10 ou pas,
 - Le constat sera :
 - « true » : oui la valeur de « note » est supérieure à 10,
 - « false » : non la valeur de « note » est inférieure ou égale à 10,
- Lignes 20 à 22 :
 - Bloc d'instructions à exécuter si le constat est « true » : c'est l'équivalent du bloc « **ALORS** » dans Un algorithme ou d'un « **Then** » de certains langages,
- Lignes 24 à 26 :
 - Bloc d'instructions à exécuter si le constat est « false » : c'est l'équivalent du bloc « **SINON** » dans Un algorithme,
- Ligne 27 :
 - Cette instruction sera exécutée, quel que soit, le constat du test if, car cette instruction est écrite en dehors du test.

Remarques :

- Si le bloc d'instructions à exécuter, dans le cas du true ou du false, ne comporte qu'une seule instruction, il est possible de ne pas mettre les accolades { },
 - Il est conseillé de mettre toujours les { }
- Deux règles du bon développeur (car il pense à ceux qui vont lire et modifier son code) :
- **Faire un code le plus explicite possible,**
 - **Faire un code le plus simple possible**
- Le bloc else n'est pas obligatoire s'il n'y a aucune action à faire dans ce cas.

Exemple 2 : comparer une variable à une autre variable

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
14 // méthode main
15 public static void main(String[] args) {
16     int nbreViesMaxi=3;
17     int nbreViesRestantes=2;
18     // .....
19     // dans une séquence de jeu vidéo
20
21     // décider du passage en année supérieure
22     if (nbreViesRestantes==nbreViesMaxi)
23     {
24         System.out.println("code pour animation de début du jeu");
25     }
26     else
27     {
28         if (nbreViesRestantes!=0)
29         {
30             System.out.println("code pour animation nouvelle vie");
31         }
32         else
33         {
34             System.out.println("code pour Game Over");
35         }
36     }
37     System.out.println("exécution de la suite du jeu");
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0090_TestsAvecIfEncore [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 09:30:31)

code pour animation nouvelle vie
exécution de la suite du jeu

Remarques :

- Avez-vous remarqué les ifs imbriqués les uns dans les autres ?

Tester plusieurs conditions

Nous avons souvent besoin de tester, dans un même test, si plusieurs conditions sont réalisées en même temps ou une à la fois,

Nous utilisons pour cela l'opérateur ET et l'opérateur OU.

Examinons le test de deux conditions dans un même test if :

Condition 1	Condition 2	Opérateur utilisé	Constat général du if
Fausse	Fausse	ET	Fausse
Fausse	Vraie	ET	Fausse
Vraie	Fausse	ET	Fausse
Vraie	Vraie	ET	Vraie
Fausse	Fausse	OU	Fausse
Fausse	Vraie	OU	Vraie

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Vraie	Fausse	OU	Vraie
Vraie	Vraie	OU	Vraie

Exemple :

Nous avons caractérisons le compte bancaire d'un client par deux variables :

- La variable « **solde** » qui contient un nombre réel positif ou négatif,
- La variable « **faciliteDeCaisse** » qui contient une valeur booléenne «true» ou «false»,

Nous souhaitons bloquer tout paiement sur le solde du client est négatif et qu'il ne bénéficie pas de facilité de caisse :

```
14 // méthode main
15 public static void main(String[] args) {
16     double solde=1275;
17     boolean faciliteDeCaisse=false;
18     //
19     // .. quelques lignes de codes plus loin
20     //
21     // bloquer ou pas le paiement
22     if (solde<0 && faciliteDeCaisse==false)
23     {
24         // instructions pour bloquer le paiement
25         System.out.println("paiement bloqué");
26     }
27     else
28     {
29         // instructions pour accepter le paiement
30         System.out.println("paiement accepté");
31     }
32     System.out.println("exécution de la suite du code")
33 }
34
```

Problems @ Javadoc Declaration Console Servers
<terminated> Cours_0100_TestsAvecIfPlusieursConditions [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (paiement accepté
exécution de la suite du code

Opérateurs de comparaison

Type de comparaison	Opérateur java	Remarque
Strictement Supérieur à	>	
Supérieur ou égal	>=	
Strictement Inférieur	<	
Inférieur ou égal	<=	
Égal	==	Pas valables pour comparer deux chaînes, À ne pas confondre avec =
Égal	Méthode equals	Valables pour deux chaînes
Différent	!=	

Opérateurs logiques

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Type de comparaison	Opérateur java	Remarque
ET	&&	
OU		Alt gr + la touche 6
NON	!	Exemple : If (!var1) {} Retourne true si var1 vaut false

ATTENTION : lorsque vous testez plusieurs conditions avec des ET et des OU : soyez conscients des Différents cas de figure en présence :

Si vous testez deux conditions : quatre cas de figure différents

Condition 1	Condition 2	Cas n°
Vraie	Vraie	1
Vraie	Fausse	2
Fausse	Vraie	3
Vraie	Vraie	4

Si vous testez trois conditions : huit cas de figure différents et non pas six

Si vous testez n conditions : 2^n cas de figure différents

Switch

Cette structure de test est une alternative au if dans le cas où :

- On teste une seule variable ;
- le nombre de valeurs à tester est connu à l'avance.

Exemple 1 : mettre une appréciation sur un bulletin de notes

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
16 int moyenne=10; // on suppose que la note de dépasse pas 20
17 //
18 // décider de l'appréciation à mettre sur le bulletin de notes
19 //
20 switch(moyenne)
21 {
22     case 0:
23         // mettre sur le bulletin que c'est une catastrophe
24         System.out.println("la cata");
25         break;
26     case 10:
27         // mettre sur le bulletin que c'est juste correct
28         System.out.println("passable");
29         break;
30     case 20:
31         // mettre sur le bulletin que c'est excellent
32         System.out.println("excellent");
33         break;
34     default:
35         // mettre sur le bulletin que c'est encourageant
36         System.out.println("encourageant : continuez vos efforts");
37         break;
38 }
39 System.out.println("exécution de la suite du code");
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0110_TestsAvecSwitch [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 09:51:46)

passable

exécution de la suite du code

- Ligne 20 : la variable à tester «**moyenne**» est citée une seule fois dans switch
- Lignes 22 à 25 : ce bloc est appelé une languette
Chaque languette doit représenter un test de valeur différente de la variable «moyenne»
- L'instruction **break** : cette instruction est obligatoire à la fin de chaque languette,
Essayez le code ci-dessus sans les breaks et constatez la bizarrerie.
- Ligne 34 : l'instruction **default** permet de traiter les autres cas différents des cas cités
Dans les languettes précédentes.

Exemple 2 : décider quelle salutation adresser à quelqu'un

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
14 // méthode main
15 public static void main(String[] args) {
16     String salutations="jour";
17     // .....
18     switch(salutations)
19     {
20         case "jour":
21             System.out.println("Bonjour Monsieur");
22             break;
23         case "soir":
24             System.out.println("Bonsoir Monsieur");
25             break;
26         case "nuit":
27             System.out.println("Bonne nuit Monsieur");
28             break;
29         default:
30             System.out.println("Quelle est heure est il Monsieur ?");
31             break;
32     }
33     System.out.println("exécution de la suite du code");
34 }
35 }
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0120_TestsAvecSwitchEncore [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 09:57:20)

Bonjour Monsieur
exécution de la suite du code

Réaliser des boucles

Comme dans tous les autres langages, Java permet d'exécuter un ensemble d'instructions n fois grâce aux boucles.

Les 3 structures principales de boucles sont :

Type de structure	Utilisation courante
La boucle for	À utiliser quand on connaît avant la boucle le nombre de répétitions (d'itérations à faire)
La boucle while	À utiliser quand le nombre de répétitions est variable car il dépend du test d'une condition et il se peut que l'on ne rentre pas du tout dans la boucle.
La boucle do while	À utiliser quand le nombre de répétitions est variable car il dépend du test d'une condition et on souhaite entrer au moins une fois dans la boucle.

For

Exemple : dire bonjour 10 fois avec une boucle for

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
8      */
9      // méthode main
10     public static void main(String[] args) {
11         // Boucle for simple
12         for(int compteur=1;compteur<11;compteur++)
13         {
14             System.out.println("Bonjour du for "+compteur+" fois");
15         }
16
17         System.out.println("exécution de la suite du code");
18     }
19 }
20
```

<terminated> Cours_0130_boucleAvecFor [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 10:29:33)

```
Bonjour du for 1 fois
Bonjour du for 2 fois
Bonjour du for 3 fois
Bonjour du for 4 fois
Bonjour du for 5 fois
Bonjour du for 6 fois
Bonjour du for 7 fois
Bonjour du for 8 fois
Bonjour du for 9 fois
Bonjour du for 10 fois
exécution de la suite du code
```

Explications sur le code ci-dessus :

- Ligne 12 :
 - on déclare une variable (ici nommée compteur) qui va servir à compter le nombre de répétitions,
 - on fixe la valeur initiale de cette variable compteur : ici fixée à 1,
 - on donne la condition à vérifier pour continuer à faire des répétitions : ici tant que la variable compteur reste inférieure à 11,
 - on dit à Java de combien faut-il incrémenter la variable compteur : ici le ++ signifie qu'il faut incrémenter la variable compteur de +1
- lignes 13 à 15 :
 - lignes de codes à exécuter n fois,
- à partir de la ligne 16 :
 - on n'est plus dans la boucle

Simulons l'exécution du code ci-dessus en mémoire :

Numéro de ligne	Valeur de la variable compteur	Condition compteur < 11	Affichage
12	1	False	
13			
14			Bonjour du for 1 fois
15	1+1 = 2	False	
13			

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

14			Bonjour du for 2 fois
15	$2+1 = 3$	False	
13			
14			Bonjour du for 3 fois
15	$3+1 = 4$	False	
13			
14			Bonjour du for 4 fois
15	$4+1 = 5$	False	
13			
14			Bonjour du for 5 fois
15	$5+1 = 6$	False	
13			
14			Bonjour du for 6 fois
15	$6+1 = 7$	False	
13			
14			Bonjour du for 7 fois
15	$7+1 = 8$	False	
13			
14			Bonjour du for 8 fois
15	$8+1 = 9$	False	
13			
14			Bonjour du for 9 fois
15	$9+1 = 10$	False	
13			
14			Bonjour du for 9 fois
15	$10+1 = 11$	True	
16	On sort de la boucle		

Remarque importante :

Dans le bloc de la boucle for : le développeur ne doit pas tenter de changer la valeur de la variable qui sert de compteur : sinon on risque soit de sortir plutôt de la boucle, soit faire une boucle infinie.

While

Exemple : dire bonjour 10 fois avec une boucle while

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
9      // méthode main
10     public static void main(String[] args) {
11         // Boucle while
12         int i=1;
13         while(i<11)
14         {
15             System.out.println("Bonjour du while "+i+" fois");
16             // attention : si je n'incrmente pas i : boucle infinie
17             i++; // mettre +1 dans i
18         }
19         System.out.println("exécution de la suite du code");
20     }
21 }
22
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0140_boucleAvecWhile [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 10:37:49)

```
Bonjour du while 1 fois
Bonjour du while 2 fois
Bonjour du while 3 fois
Bonjour du while 4 fois
Bonjour du while 5 fois
Bonjour du while 6 fois
Bonjour du while 7 fois
Bonjour du while 8 fois
Bonjour du while 9 fois
Bonjour du while 10 fois
exécution de la suite du code
```

Remarques importantes :

- Ici le développeur doit faire en sorte que la condition devienne, à un moment, fausse, pour sortir de la boucle et donc ne pas faire une boucle infinie
- Si dans la ligne 12 : je mets par exemple 20 dans i : on n'entrera jamais dans la boucle. Donc le while est Fait pour cela : on peut rentrer ou pas dans la boucle.

Do while

Exemple : dire bonjour 10 fois avec une boucle do while

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
12  int i=14;
13
14  do
15  {
16      System.out.println("Bonjour du do while "+i+" fois");
17      // attention : si je n'incrémente pas i : boucle infinie
18      i++; // mettre +1 dans i
19  } while(i<11);
20  System.out.println("exécution de la suite du code");
21  }
22 }
23
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0150_boucleAvecDoWhile [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 10:45:21)

Bonjour du do while 14 fois
exécution de la suite du code

Pourquoi, ci-dessus, la boucle se fait une et une seule fois ? Et pourquoi se fait-elle alors que la condition est fausse ?

Opérateurs de calcul

Pour effectuer des calculs, Java met à notre disposition les opérateurs suivants :

Opérateur	Nom	Rôle	Exemple	Résultat
+	Opérateur d'addition	Additionner 2 valeurs numériques	int z=6 ; z=z+2 ;	z=8
-	Opérateur de soustraction	Soustraire une valeur d'une autre valeur	int z=6 ; z=z-3 ;	z=6
*	Opérateur de multiplication	Multiplier une valeur par une autre valeur	int z=6 ; z=z * 2 ;	z=12
/	Opérateur de division	Diviser une valeur par une autre valeur	int z= 6 ; int d=3 ; z=z/d ;	z=2 ; d=3 ;
%	Opérateur division euclidienne	Récupère le reste d'une division euclidienne	int z=6 ; int reste=z%4 ;	Reste=2 ; 6 divisé par 4 donne un résultat entier de 1 et un reste de 2.
=	Opérateur d'affectation	Remplit une variable par une valeur	int z=6 ; z=27 ;	Z=27 ;

ATTENTION : pour tester égalité numérique avec un if : on utilise == et pas =

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Opérateurs d'assignation

Pour simplifier l'ajout, la soustraction, la multiplication ou la division d'une valeur à une variable.

Par exemple : si je veux ajouter 2 à une variable f :

Soit je fais `f=f+2 ;`

Soit je fais `f+=2 ;`

Opérateur	Effet
<code>+=</code>	addition deux valeurs et stocke le résultat dans la variable (à gauche)
<code>-=</code>	soustrait deux valeurs et stocke le résultat dans la variable
<code>*=</code>	multiplie deux valeurs et stocke le résultat dans la variable
<code>/=</code>	divise deux valeurs et stocke le quotient dans la variable
<code>%=</code>	divise deux valeurs et stocke le reste dans la variable

Opérateurs d'incrémentation

Si je souhaite simplement ajouter ou soustraire 1 d'une variable je peux utiliser ++ ou --.

Exemple : ajouter 1 à la variable y :

Soit `y=y+1 ;`

Soit `y++ ;`

Opérateur	Dénomination	Effet	Syntaxe	Résultat (int x=7)
<code>++</code>	Incrémentatation	Augmente d'une unité la variable	<code>x++</code> ou <code>++x</code>	8
<code>--</code>	Décrémentatation	Diminue d'une unité la variable	<code>x--</code> ou <code>--x</code>	6

On met le ++ avant ou après la variable : on n'obtient pas le même résultat : Essayez.

Manipuler des chaînes de caractères

Les variables déclarées de type String peuvent accéder à des méthodes fournies par Java et qui servent à manipuler les chaînes de caractères.

Exemple :

J'ai une variable de type String et nommée «nom» ,

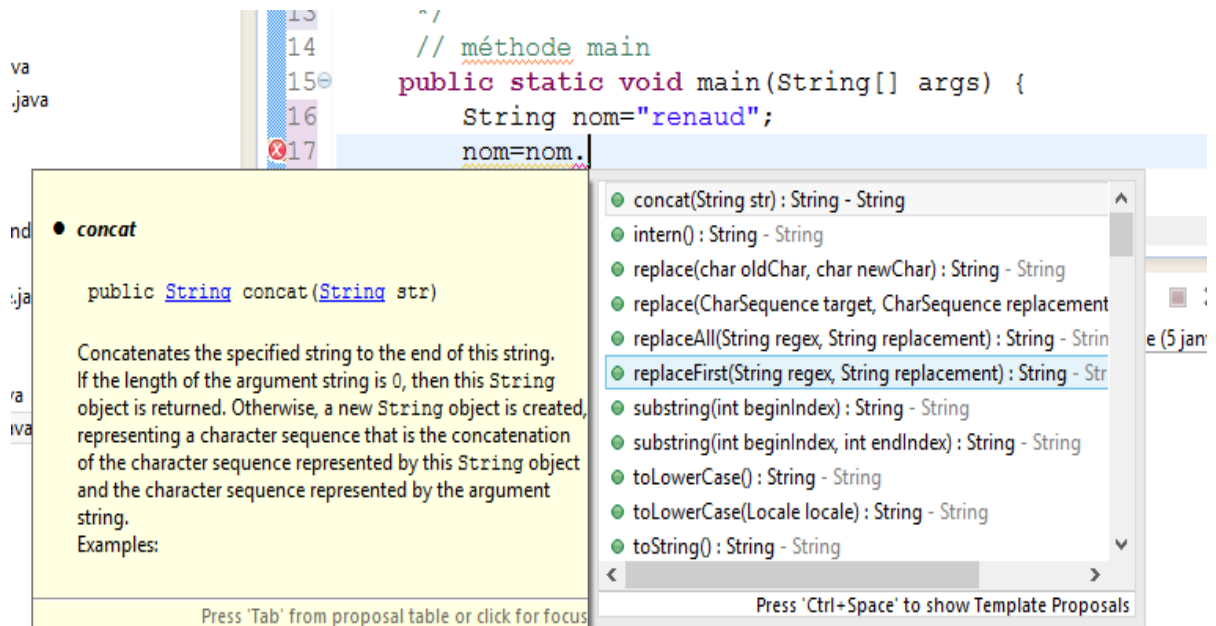
Je souhaite mettre ce nom en majuscule ou en minuscule,

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Au lieu de faire tout un algorithme pour tenter par mes propres moyens de faire cette manipulation, java me fournit le nécessaire :



Je tape le nom de ma variable de type String puis je tape « . » : Eclipse m'affiche toutes les méthodes fournies par Java pour manipuler les chaînes de caractères.

Citons quelques méthodes très utiles :

Nom méthode pour manipuler une chaîne	Rôle
concat	Ajouter à la fin de la chaîne une autre chaîne
replace	Remplacer un caractère ou une sous chaîne dans la chaîne
substring	Extraire une sous chaîne de la chaîne
toLowerCase	Mettre en minuscules la chaîne
toUpperCase	Mettre en majuscules la chaîne
.... Etc	

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
14 // méthode main
15 public static void main(String[] args) {
16     String nom="rEnaud";
17     System.out.println("nom d'origine : "+nom);
18     nom=nom.toLowerCase();
19     System.out.println("nom en minuscules : "+nom);
20     nom=nom.toUpperCase();
21     System.out.println("nom en majuscules : "+nom);
22 }
23 }
24
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0160_manipulationChaines [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. ...)

```
nom d'origine : rEnaud
nom en minuscules : renaud
nom en majuscules : RENAUD
```

Conversion de types

Il va vous arriver souvent de vouloir mettre dans une variable une valeur qui ne serait pas appropriée.

Par exemples :

- Mettre un entier dans une variable de type décimal (float ou double),
- Mettre un décimal dans une variable de type entier,
- Mettre un numérique contenu dans une chaîne dans une variable de type entier,
-etc

Java (ou plus exactement **la machine virtuelle**) :

- Examine chaque affectation pour savoir s'il y a besoin de convertir ou pas le type d'une valeur,
- Si oui :
 - Elle évalue si elle peut faire la conversion directement : cela s'appelle une conversion implicite (Sans perte de données ou perte de précision),
 - Si elle ne peut pas convertir directement (car il y aura risque de perte de données ou il est impossible de convertir) : elle oblige le développeur à préciser dans quel type faut-il convertir : Cela s'appelle une conversion explicite.

Exemples :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
15 public static void main(String[] args) {
16     // déclaration de variables
17     int a=120;
18     double b=0;
19     double c=128.13;
20     int z;
21     String q="123";
22     String s="blabla";
23     char h='A';
24     // conversion implicite : accepté par Vm
25     b=a; // je mets un entier dans un décimal : il n'y a pas de perte
26         // de précision
27     //
28     // z=c; // cette ligne est refusée : java dit qu'il ne peut pas convertir
29         // en Anglais : type mismatch .....
30     //
31     z=(int) c; // le développeur demande une conversion explicite
32         // z contiendra 128 : on perd 13 cents : c'est un choix
33         // du développeur
34     z=Integer.parseInt(q); // le développeur demande une conversion explicite de String
35         // vers int : ici ça va : la chaîne contient un entier
36     //z=Integer.parseInt(s); // au codage : pas d'erreur mais à l'exécution : il y aura
37         // plantage : on ne peut pas convertir 'blabla' en un entier
38 }
```

Problems Javadoc Declaration Console Servers

<terminated> Cours_0170_conversionDetypes [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 11:53:19)

z : 65

Utiliser des tableaux

Tous les langages de programmation manipulent des tableaux,

Nous avons besoin d'utiliser un tableau, lorsqu'il faut stocker plusieurs valeurs de même nature :

- stocker les noms de plusieurs clients, les notes de plusieurs étudiants, les emails de plusieurs Personnes,etc.

Exemple 1 : Tableau à une dimension contenant plusieurs noms :

Dupont	Durand	Lefaire	Jacquet	benarfa
--------	--------	---------	---------	---------

- Le tableau portera un **nom unique**,
- Chaque case du tableau est appelé « **élément** »,
- Chaque élément a une **position** dans le tableau : le 1^{er} élément a la position 0, le 2^{ème} élément a la position 1,
- Le tableau a un **type** : les éléments du tableau sont de type int ou String ou char ou

En java :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
17  /* =====
18  *  déclaration un tableau simple avec initialisation :
19  *  il n'y a plus besoin de donner le nombre d'éléments
20  *  du tableau : java a calculé qu'il y a 5 éléments
21  *  =====
22  */
23  String [] lstNoms1={"Dupont","Durand","Lefaure","Jacquet","benarfa"};
24  // j'affiche le premier élément du tableau (élément numéro 0)
25  System.out.println("nom : "+lstNoms1[0]);
26  // j'affiche le dernier élément du tableau (élément numéro 4)
27  System.out.println("nom : "+lstNoms1[4]);
28  /* =====
29  *  déclaration un tableau simple sans initialisation :
30  *  je dois préciser le nombre d'éléments
31  *  =====
32  */
33  String [] lstNoms2=new String [5];
34  // ... quelques lignes plus loin : je remplis le tableau
35  lstNoms2[0]="Dupont";
36  lstNoms2[1]="Durand";
37  lstNoms2[2]="Lefaure";
38  lstNoms2[3]="Jacquet";
39  lstNoms2[4]="benarfa";
40  // j'affiche l'élément 3 du tableau
41  System.out.println("nom : "+lstNoms2[2]);|
```

Problems @ Javadoc Declaration Console Servers
<terminated> Cours_0180_TabelauSimple [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 12:56:34)
nom : Dupont

Explications :

- Ligne 23 : déclaration d'un tableau à une dimension avec initialisation
- Ligne 33 : déclaration d'un tableau à une dimension sans initialisation (existe mais vide)
- Lignes 25, 27, 35 à 41 : utilisation des éléments d'un tableau.

Exemple 2 : Tableau à deux dimensions contenant 7 lignes correspondant à 7 jours de la semaine et Une température relevée par jour :

1	24
2	18
3	19
4	28
5	17
6	21
7	20

En java :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

```
15 public static void main(String[] args) {
16
17     /* =====
18     *  déclaration un tableau à deux dimensions contenant des entiers
19     *  =====
20
21     7 lignes      : 1 ligne par jour de semaine
22     2 colonnes   :  une colonne pour le n° de jour,
23                   une colonne pour température relevée
24
25     */
26     int [][] lstTempSemaines={1,24},{2,18},{3,19},{4,28},{5,17},{6,21},{7,20}};
27     // .. quelques lignes de code plus loin
28     // Boucle d'affichage de température
29     for(int i=0;i<7;i++)
30     {
31         int numJour=lstTempSemaines[i][0];
32         switch(numJour)
33         {
34             case 1:
35                 System.out.println("Température du Lundi : "+lstTempSemaines[i][1]);
36                 break;
37             case 2:
38                 System.out.println("Température du Mardi : "+lstTempSemaines[i][1]);
39                 break;
40             case 3:
41                 System.out.println("Température du Mercredi : "+lstTempSemaines[i][1]);
42                 break;
43             case 4:
44                 System.out.println("Température du Jeudi : "+lstTempSemaines[i][1]);
45                 break;
46             case 5:
47                 System.out.println("Température du Vendredi : "+lstTempSemaines[i][1]);
48                 break;
49             case 6:
50                 System.out.println("Température du Samedi : "+lstTempSemaines[i][1]);
51                 break;
52             case 7:
53                 System.out.println("Température du Dimance : "+lstTempSemaines[i][1]);
54                 break;
55         }
56     }
57 }
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0190_TabelauDeuxDimensions [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 15:54:22)

```
Température du Lundi : 24
Température du Mardi : 18
Température du Mercredi : 19
Température du Jeudi : 28
Température du Vendredi : 17
Température du Samedi : 21
Température du Dimance : 20
```

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Gestion des exceptions

Dans un code, deux types d'erreurs peuvent se produire :

- Des **erreurs lors de la saisie** : Eclipse signale ces erreurs et elles doivent être modifiées avant de pouvoir tester le code,
- Des **erreurs d'exécution** : elles se produisent pendant l'exécution et souvent interrompent anormalement l'application,
 - Ces erreurs sont difficiles à détecter et demandent une attention particulière de la part du développeur,

Exemple d'erreurs à l'exécution

```
14 // méthode main
15 public static void main(String[] args) {
16     int a=12,b=0,c;
17     char [] alphabet=new char[26];
18     // erreur de division par zéro
19     c=a/b;
20     // tentative d'accéder à un élément hors de portée d'un tableau
21     alphabet[27]='z';
22 }
23 }
```

Problems @ Javadoc Declaration Console Servers

<terminated> Cours_0200_gestionExceptions [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 16:22:30)

Exception in thread "main" java.lang.ArithmeticException: / by zero
at pack_10_Cours.Cours_0200_gestionExceptions.main(Cours_0200_gestionExceptions.java:19)

Explications sur le code ci-dessus :

- Ligne 9 : cette instruction va diviser par 0 : impossible mathématiquement,
- Ligne 21 : cette instruction tente d'accéder au 27^{ème} élément d'un tableau dont Les éléments sont indexés de 0 à 25,

Exécution :

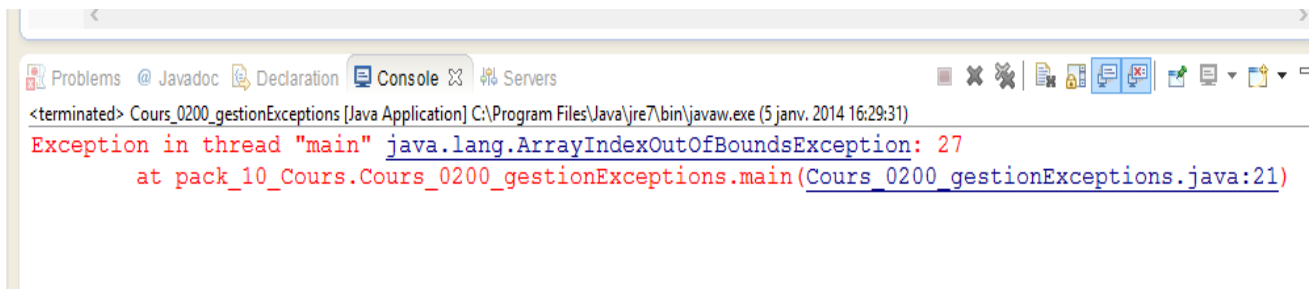
- Ligne 9 : provoque l'arrêt du code avec le message d'erreur en bas du code : On parle de **ArithmeticException**
- Ligne 21 : on met en commentaire la ligne 9 pour qu'elle ne bloque plus le programme et on ré exécute le programme,

Cela donne un message erreur :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



```
<terminated> Cours_0200_gestionExceptions [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 16:29:31)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 27
    at pack_10_Cours.Cours_0200_gestionExceptions.main(Cours_0200_gestionExceptions.java:21)
```

On parle de **ArrayIndexOutOfBoundsException**

Les exceptions

Java appelle ces erreurs d'exécution : des **Exceptions**,

Chaque exception a un nom : `ArithmeticException`, `ArrayIndexOutOfBoundsException`,

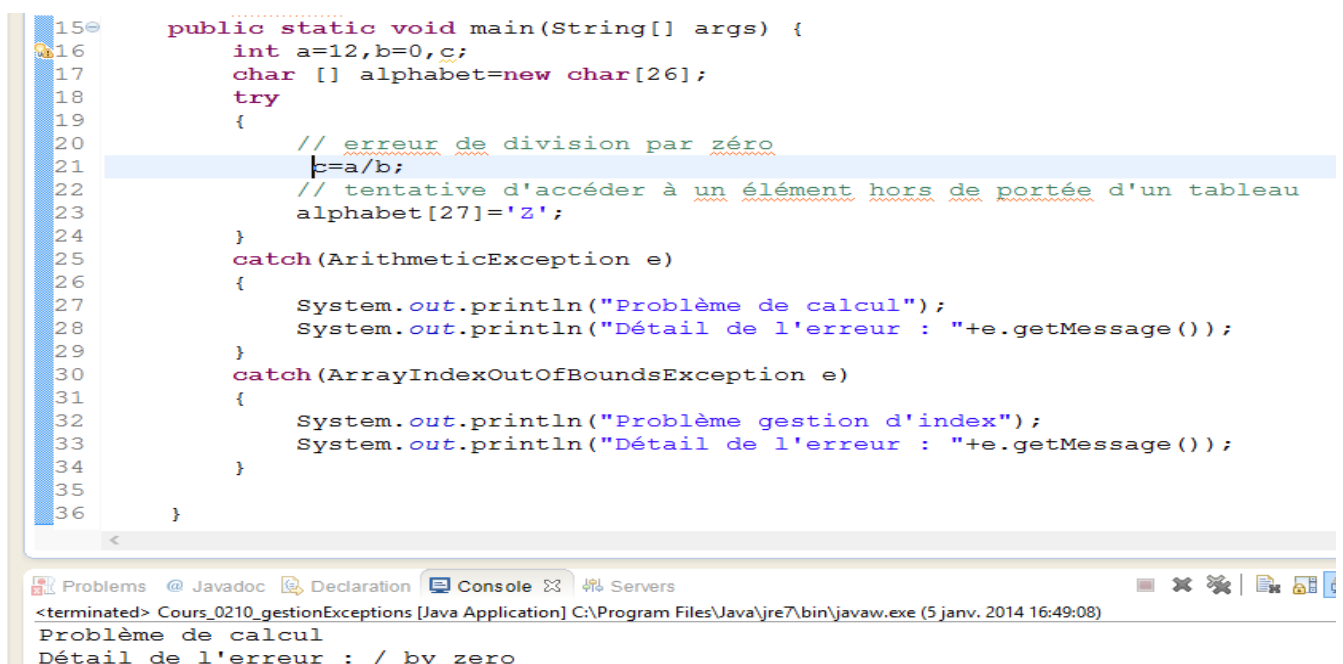
Gérer les Exceptions

Deux situations obligent le développeur à gérer les exceptions (c'est-à-dire : faire en sorte que le programme ne s'arrête pas anormalement) :

- Le développeur sait qu'une situation d'exception peut se produire : une division par zéro peut arriver, une connexion à une BD peut échouer, un accès à un serveur réseau devient impossible, ...
- Certaines instructions Java exigent du développeur qu'il gère des exceptions que ces mêmes instructions peuvent produire,

Comment gérer les Exceptions ?

Le développeur doit mettre en place un gestionnaire d'exceptions ainsi :



```
15 public static void main(String[] args) {
16     int a=12,b=0,c;
17     char [] alphabet=new char[26];
18     try
19     {
20         // erreur de division par zéro
21         f=a/b;
22         // tentative d'accéder à un élément hors de portée d'un tableau
23         alphabet[27]='Z';
24     }
25     catch(ArithmeticException e)
26     {
27         System.out.println("Problème de calcul");
28         System.out.println("Détail de l'erreur : "+e.getMessage());
29     }
30     catch(ArrayIndexOutOfBoundsException e)
31     {
32         System.out.println("Problème gestion d'index");
33         System.out.println("Détail de l'erreur : "+e.getMessage());
34     }
35 }
36 }
```

```
<terminated> Cours_0210_gestionExceptions [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 16:49:08)
Problème de calcul
Détail de l'erreur : / by zero
```

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Explications sur le code ci-dessus ;

- Lignes 18 à 24 : on encadre le code susceptible de produire une exception par un Bloc **try { instructions }**
- Lignes 25 à 29 : le bloc **catch** sert à intercepter une exception et coder les Instructions pour gérer cette erreur,
Ce 1^{er} catch veut gérer l'exception **ArithmeticException**
- Lignes 30 à 34 : le bloc **catch** sert à intercepter une exception et coder les Instructions pour gérer cette erreur,
Ce 2^{ème} catch veut gérer l'exception **ArrayIndexOutOfBoundsException**

Remarques :

- Si une exception autre que celles gérer par les catches se produit : le programme plante,
- Si vous souhaitez gérer les autres types d'exceptions sans les nommer une par une : il faut gérer (catcher) l'exception qui se nomme simplement **Exception**,
- Un bloc **finally**, optionnel, peut être mis après le try et le catch, et il sert à exécuter un code De fin de gestionnaire d'exceptions : qu'il y ait exception ou pas : ce code sera exécuté.
Exemple d'utilisation de finally : procéder à des fermetures de fichiers ou à des déconnexions,etc.

Les méthodes de classes

On rappelle qu'une **méthode** est l'équivalent de **fonction** ou **procédure** dans les langages non orientés objets, Jusqu'à maintenant, nous allons utiliser une et une seule méthode, la méthode «**main**» dans nos différents exemples de code,

Or souvent le code dans une classe est répartie sur plusieurs méthodes pour :

- Spécialiser des portions de codes,
- Faciliter la maintenance de code volumineux,
- Réutiliser la méthode à partir d'un code extérieur à la classe,
- Etc.

Syntaxe générale d'une méthode

Visibilité type appel type de réponse nom méthode (type param1 nom param1, type param2 nom param2,)
{
Instructions
}

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Explications :

- **Visibilité** : on met « **public** » ou « **private** » ou « **protected** » :
Dans ce 1^{er} cours d'apprentissage des bases de Java, on se contentera de mettre « **public** » sans explications.
Dans le prochain cours sur la Programmation Orientée Objets (POO), la visibilité sera expliquée,
- **Type appel** : on peut ne rien mettre ici ou mettre le mot « **static** »,
Sans connaître la POO : on met pour l'instant « **static** »
- **Type de réponse** : une méthode, lorsqu'elle est appelée par un autre code : exécute le code
Qu'elle contient et **renvoie ou pas**, au code appelant une **réponse**,
 - Si la méthode ne doit pas renvoyer de réponse : on met le mot « **void** »
À la place du type de réponse,
 - Si la méthode doit renvoyer une réponse, elle doit préciser quel type de donnée elle renvoie : **int**, **char**, **String**, **float**, **boolean**,
- **Paramètres** : une méthode peut avoir besoin d'informations pour s'exécuter,
Ces informations seront fournies par le code appelant,
On appelle ces informations : les paramètres,
Pour chaque paramètre, la méthode déclare la variable, dans laquelle,
Elle souhaite le recevoir.
- **Le code de la méthode** : il est encadré par des { }

Signature d'une méthode

On appelle la signature d'une méthode : l'ensemble des informations sur sa visibilité, son type de réponse, son nom, les paramètres qu'elle reçoit,
La signature est la 1^{ère} ligne de la méthode.

Exemples de signatures de méthodes

Signature de la méthode	Explications
public static void maMethode() { }	➔ Visibilité publique, ➔ La méthode ne renvoie pas de réponse, ➔ La méthode ne reçoit pas de paramètres
public static void maMethode(int p) { }	➔ Visibilité publique, ➔ La méthode ne renvoie pas de réponse, ➔ La méthode reçoit un paramètre et le stocke dans la variable locale p de type int
public static void maMethode(int p, char h) { }	➔ Visibilité publique, ➔ La méthode ne renvoie pas de réponse, ➔ La méthode reçoit un 1 ^{er} paramètre et le stocke dans la variable locale p de type int et un 2 ^{ème} paramètre et le stocke dans la variable locale h

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

<pre> public static String maMethode() { // return ""; } </pre>	<p>de type char</p> <ul style="list-style-type: none"> ➔ Visibilité publique, ➔ La méthode renvoie une réponse de type String, ➔ La méthode ne reçoit pas de paramètres, ➔ Une méthode qui renvoie une réponse doit contenir forcément un return
<pre> public static String [] maMethode(short longueur) { // return tabConversion; } </pre>	<ul style="list-style-type: none"> ➔ Visibilité publique, ➔ La méthode renvoie une réponse de type un tableau de chaînes (String), ➔ La méthode reçoit un paramètre et le stocke dans une variable locale longueur de type short, ➔ La variable renvoyée et nommée tabConversion est forcément un tableau de chaînes déclaré et rempli dans la méthode

Appels de méthodes localisées dans la même classe : Exemples en java

Exemple 1 : une méthode qui reçoit deux entiers et renvoie leur somme

```

14 // méthode main
15 public static void main(String[] args) {
16 //
17 // appel de la méthode addition1 en lui passant deux entiers
18 // et en receptionnant la somme dans une variable locale
19 int s=addition1(13,15); // après cet appel s contiendra 28
20 //
21
22 }
23 // méthode
24 public static int addition1(int a, int b)
25 {
26     int somme;
27     somme=a+b;
28     return somme;
29 // on aurait pu faire une seule instruction au lieu de 3 :
30 // return (a+b);
31 }
32 }

```

- Lignes 15 à 22 : méthode main,
On appelle la méthode « addition1 » et on réceptionne sa réponse dans la Variable locale (à main) nommée s,
- Lignes 24 à 31 : méthode « addition1 »,
Elle réceptionne le 1^{er} paramètre (le nombre 13) dans la variable locale a,
Et le 2^{ème} paramètre (le nombre 15) dans la variable locale b,
Elle calcule la somme, la stocke dans la variable locale nommée « somme » et la renvoie au code appelant : c'est-à-dire à la méthode main
- Lignes 20 à la fin de main : l'exécution dans la méthode continue, après l'appel de la méthode « addition1 »,

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

La méthode main a reçu le nombre 28 dans sa variable locale nommée s.

Exemple 2 : une méthode qui reçoit une chaîne et l'affiche sur la console sans rien renvoyer

```
14 // méthode main
15 public static void main(String[] args) {
16     //
17     // appel de la méthode addition1 en lui passant deux entiers
18     // et en recevant la somme dans une variable locale
19     int s=addition1(13,15); // après cet appel s contiendra 28
20     //
21     // appel de la méthode afficher
22     afficher("hello");
23     // ....
24 }
25 // méthode
26 public static int addition1(int a, int b) {
27     //
28     //
29     //
30     //
31     //
32     //
33     //
34 // méthode
35 public static void afficher(String s)
36 {
37     System.out.println(s);
38 }
39 }
```

Problems @ Javadoc Declaration Console Servers
<terminated> Cours_0220_AppelDeMéthodes [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (5 janv. 2014 18:49:07)
hello

Explications sur le code ci-dessus :

- Ligne 22 : la méthode **main** appelle la méthode « afficher » et lui passe le texte « hello »,
- Ligne 35 à 38 : la méthode «**afficher**»,
Elle ne renvoie rien,
Elle reçoit une chaîne de caractères qu'elle stocke dans sa variable locale « s »,
Dans son code, cette méthode, affiche sur la console le texte reçu.

Appel d'une méthode statique localisée dans une autre classe :

Si je suis dans la classe C1 et dans la méthode M1 de cette classe et que je souhaite appeler la méthode statique M2 de la classe C2, je procède ainsi :

C2.M2(....)

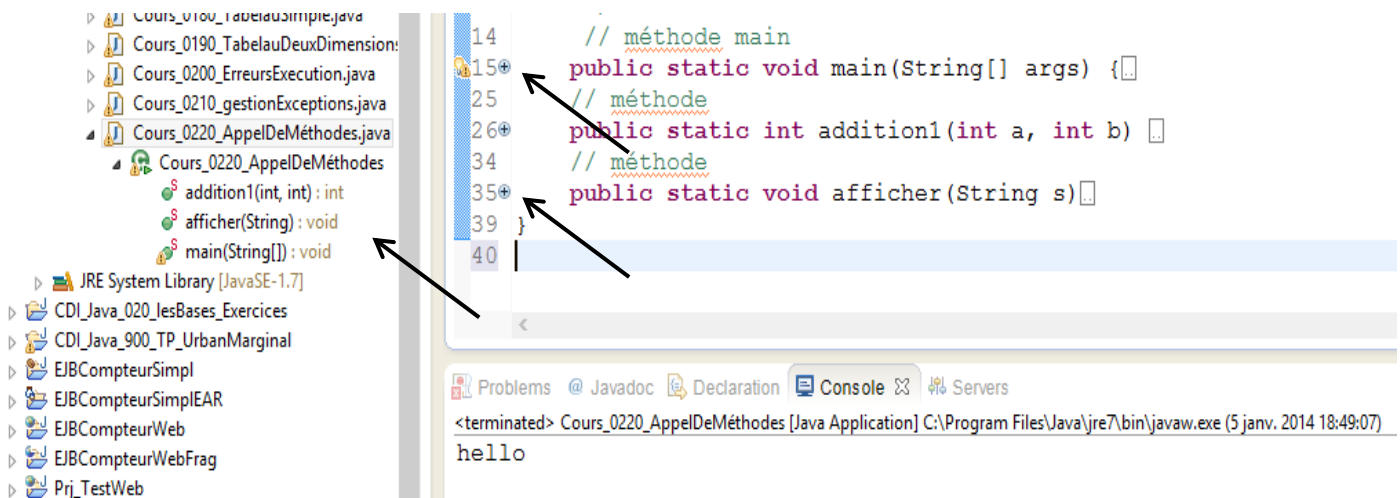
Une astuce pour organiser les méthodes

Lorsqu'une classe contient plusieurs méthodes, il peut devenir difficile de s'y retrouver,
Souvent, nous n'avons pas besoin de voir le code de toutes les méthodes en même temps,
Regardez ceci :

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



On voit ci-dessus :

- À gauche : d'un clin d'œil, je vois la liste des méthodes de la classe et leurs signatures,
- À droite : grâce au signe + et – à gauche de chaque méthode, je peux montrer ou Masquer le code d'une méthode.

Débogage de l'application

Un développeur est confronté un jour à un **programme qui «plante»** pour une raison apparemment Inexpliquée.

Dans ce cas, il y a plusieurs solutions :

- Passer en **revue le code** en espérant trouver l'erreur,
- Garnir le code d'affichages divers et variés par l'instruction «**System.out.println(..)**» afin de tracer son Exécution et de déterminer la ligne ou la méthode qui pose problème.
Bien que largement utilisée, cette solution est fastidieuse et lorsque le problème est résolu, il faut ensuite supprimer toutes les instructions d'affichage,
- Utiliser un débogueur qui est un outil, inclus dans Eclipse, qui exécute le code et permet de suspendre son Exécution, la reprendre, afficher le contenu de variables, etc.

C'est cette dernière solution que nous allons étudier avec le **débogueur intégré à Eclipse**.

Plusieurs façons d'utiliser le débogueur d'Eclipse. Nous allons initier une utilisation simple.

À vous d'approfondir l'utilisation au fur et à mesure de vos apprentissages.

Placer de points d'arrêts

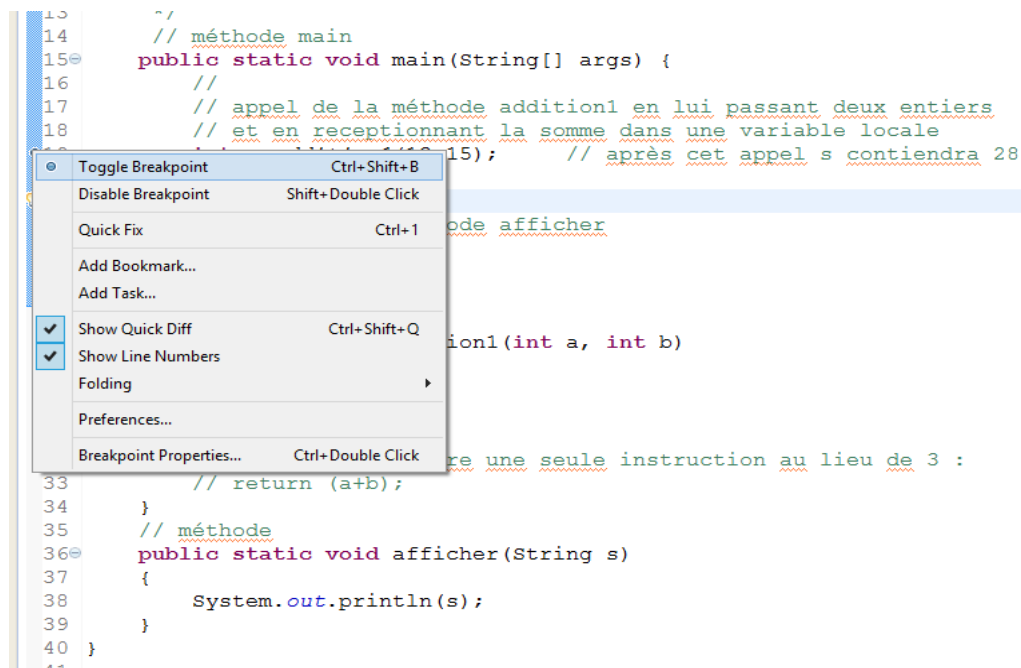
Un point d'arrêt ou **BreakPoint** est une demande de suspension de l'exécution du programme à une certaine ligne afin d'examiner le contenu des variables et voir comment le programme évolue,

Cliquez doit sur la bande grise, à gauche d'une instruction et choisissez l'option «Toggle BreakPoint» :

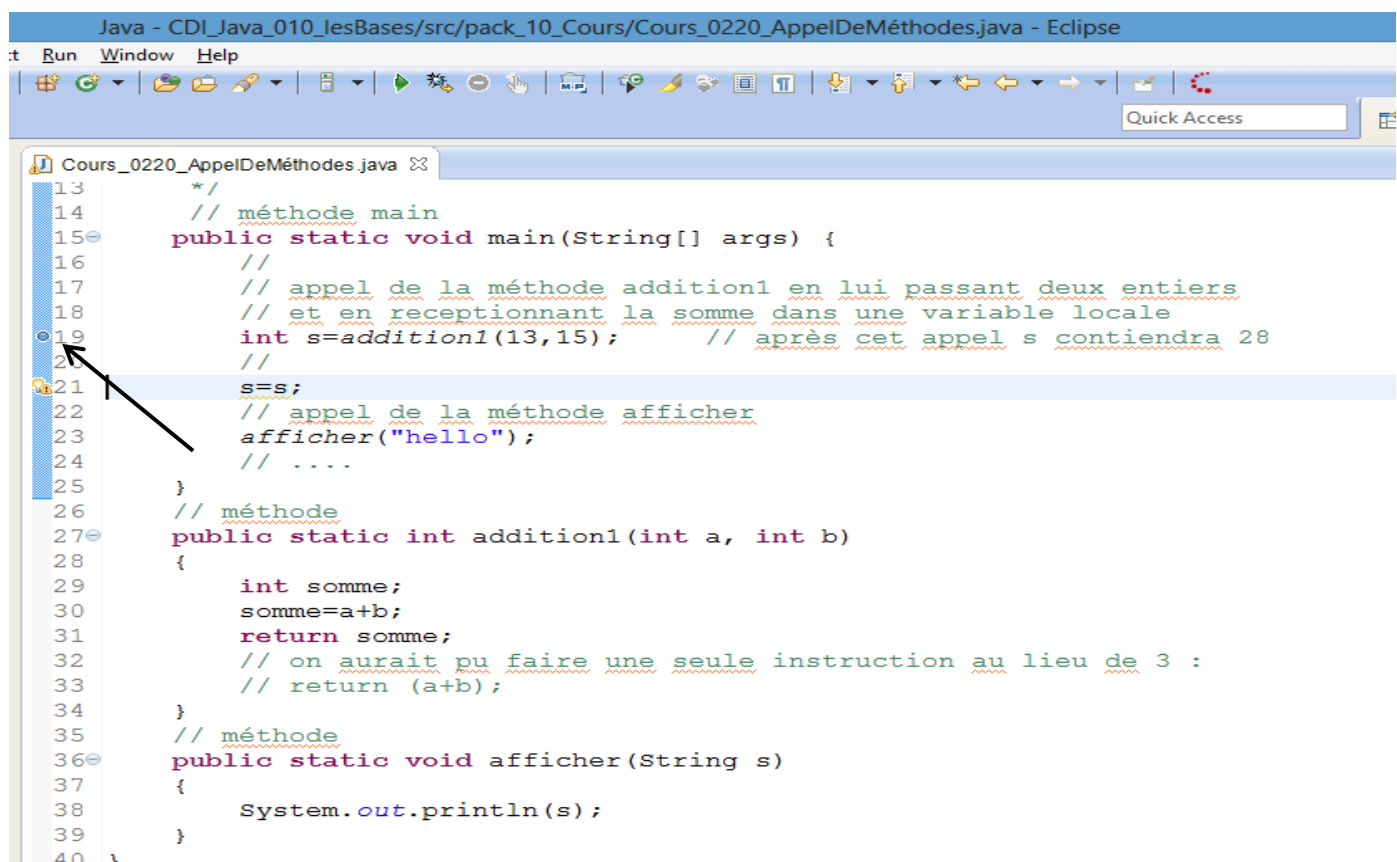
Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage



Un point bleu s'affiche à gauche de la ligne :



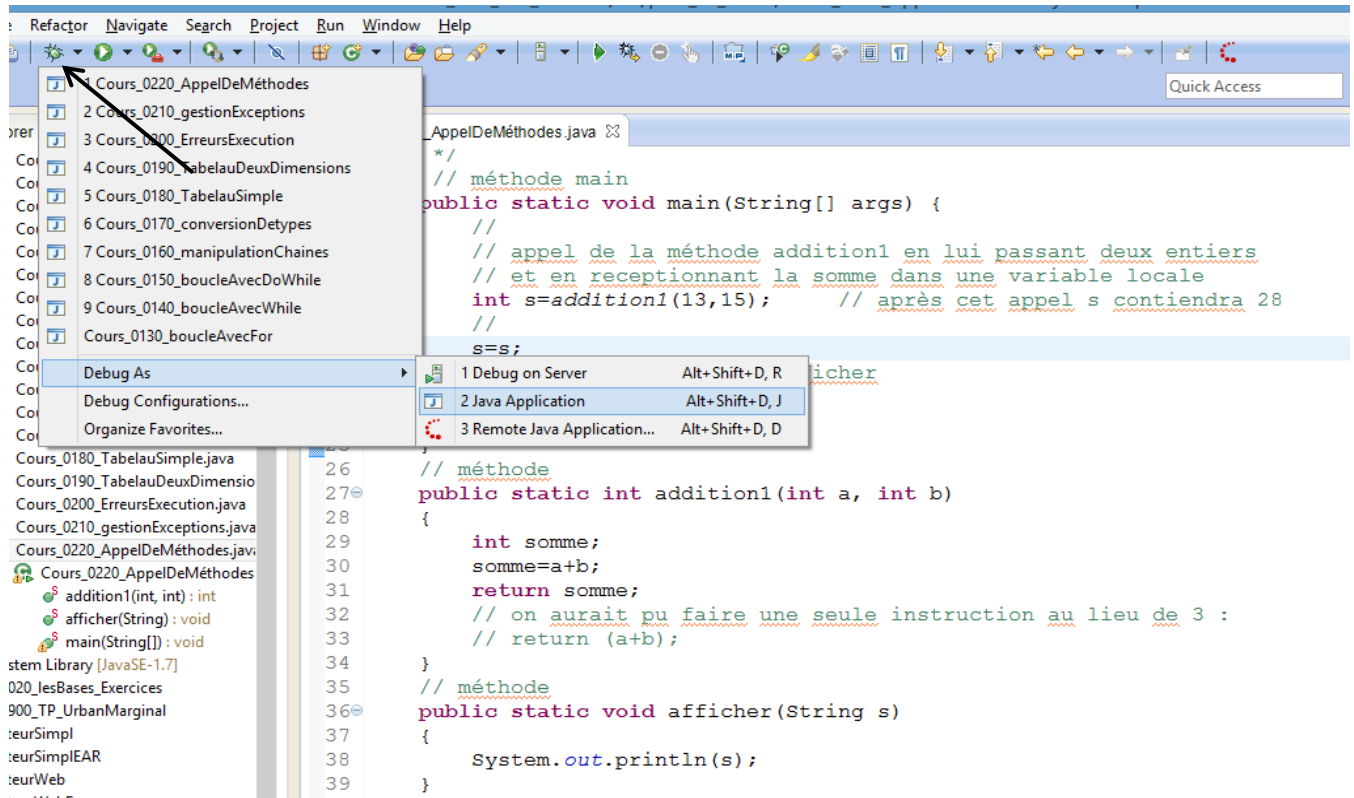
Concepteur Développeur Informatique

Module : Programmation Java

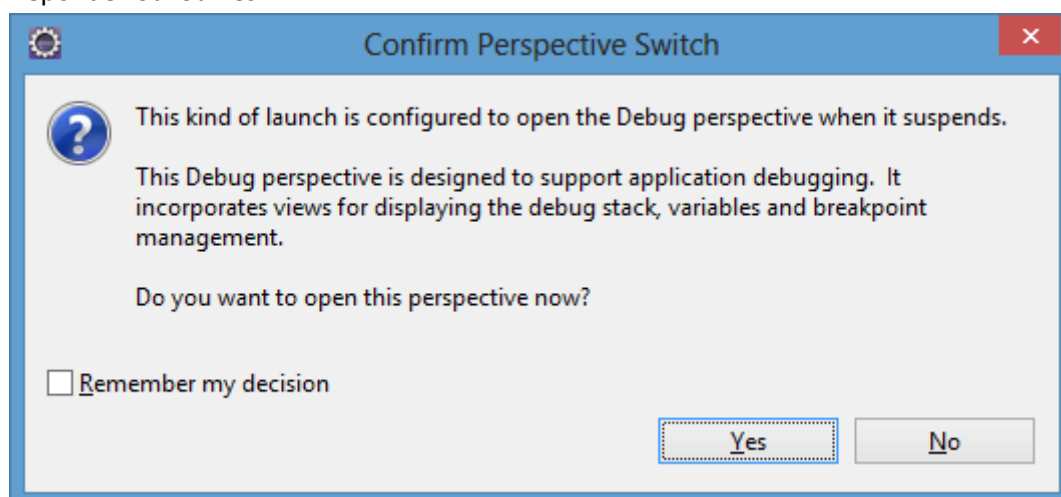
Bases du Langage

Lancer le débogage

Cliquez sur l'icône :



Une fenêtre vous avertit que l'affichage dans Eclipse va changer (changement de perspective) :
Répondez oui ou Yes :

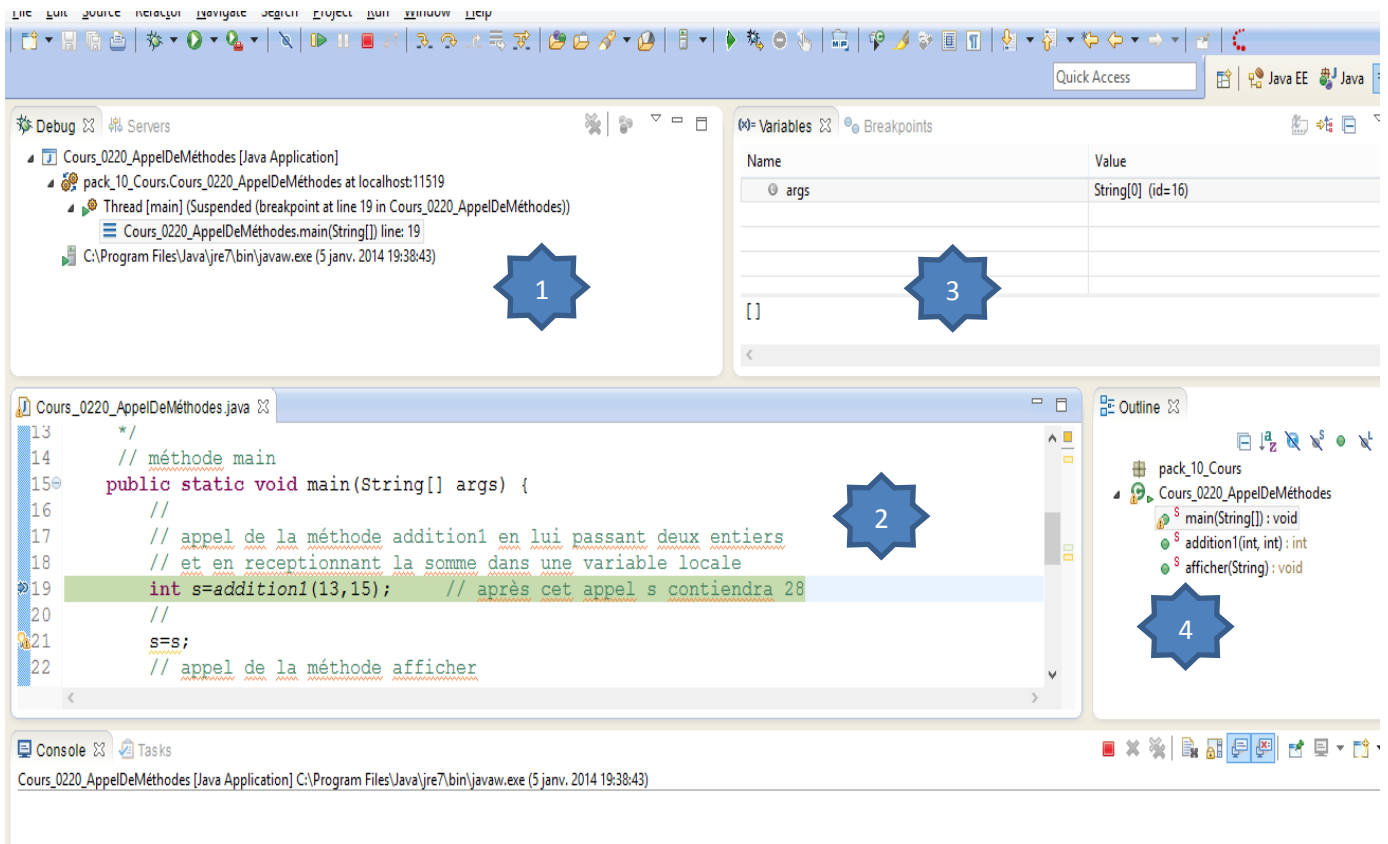


La perspective du Debug donne ceci :

Concepteur Développeur Informatique

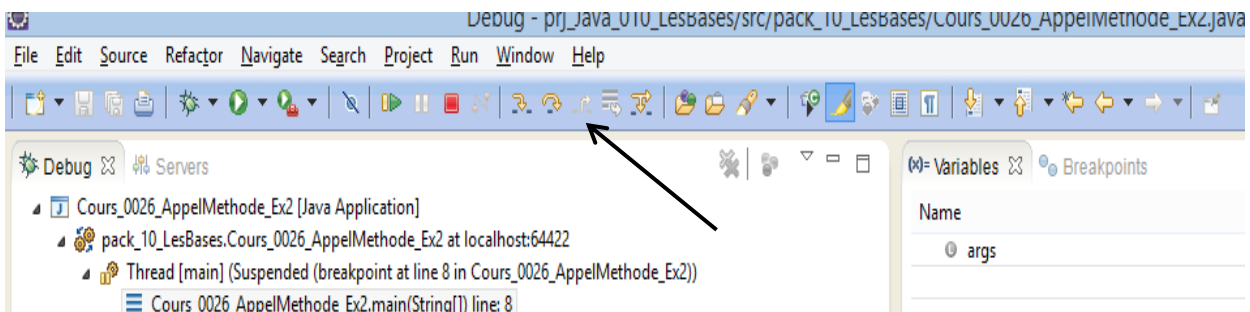
Module : Programmation Java

Bases du Langage



- Vous remarquez que l'on a changé de présentation (de Perspective) dans Eclipse : tout en haut à droite, vous avez des boutons pour passer d'une perspective à une autre : Debug, Java, JavaEE, ...
- Le volet 1 ci-dessus, vous rappelle le projet que vous êtes en train de débbuger,
- Le volet 2 vous montre le code source : la ligne sur laquelle s'arrête l'exécution est en couleur verte claire,
- Le volet 3 montre le contenu des variables au fur et à mesure de l'exécution,
- Le volet 4 montre, au fur et à mesure, de l'exécution, les classes et les méthodes.

Vous avez, en haut, dans la barre des icônes, des icônes qui représentent des flèches jaunes :



Approchez la souris de chacune de ces icônes afin de trouver celle qui affiche « Step in » : cette icône nous permet d'exécuter l'instruction courante et attendre à la suivante.

Essayez et observez en même temps le contenu des variables dans le volet 3.

Concepteur Développeur Informatique

Module : Programmation Java

Bases du Langage

Quand vous avez fini le débogage : cliquez sur l'icône du carré rouge pour stopper le débogage, fermez la perspective et revenez à la saisie du code.

Cette Deuxième partie d'apprentissage des bases de Java est terminée.