Développer une application en Java

Dr-Ing Pierre-Emmanuel Gros
Responsable Pôle Innovation Neuresys

Pierre-emmanuel.gros@neuresys.fr

Introduction à la programmation avec Java

- Introduction
- Les constructions de base d’un programmes
- Tests Logiciels
- Bonne Pratiques de conception d’une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d’un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Introduction à la programmation avec Java

- Introduction
  - Qu'est-ce que Java, une JVM
  - Le framework Java EE, les différents serveurs Applicatif J2EE ?
  - Qu'est-ce qu'Eclipse ? Comment le configurer ?
  - La vue JAVA/J2EE
  - Le debuggage
- Les constructions de base d'un programmes
- Tests Logiciels
- Bonne Pratiques de conception d'une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d'un projet avec Maven
- Mesure de la qualité

Les fondements de la programmation

Qu'est-ce que Java, une JVM

- Contenu
  - Savoir ce qu’est Java?
  - Qu’est ce qu’une JVM

Java

- Java est un langage de programmation
  - Voir le « white paper » de J.Gosling
  - Un programme Java est compilé et interprété

- Java est une plateforme
  - La plateforme Java, uniquement software, est exécutée sur la plateforme du système d'exploitation
  - La« Java Platform » est constituée de :
    - La « Java Virtual Machine » (JVM)
    - Des interfaces de programmation d'application (Java API)

Points particuliers

Java est un langage de programmation particulier qui possède des caractéristiques avantageuses:

- Simplicité et productivité:
  - Intégration complète de l'OO
  - Gestion mémoire (« Garbage collector »)
- Robustesse, fiabilité et sécurité
- Indépendance par rapport aux plateformes
- Ouverture:
  - Support intégré d'Internet
  - Connexion intégrée aux bases de données (JDBC)
  - Support des caractères internationaux
- Distribution et aspects dynamiques
- Performance

Java/C++

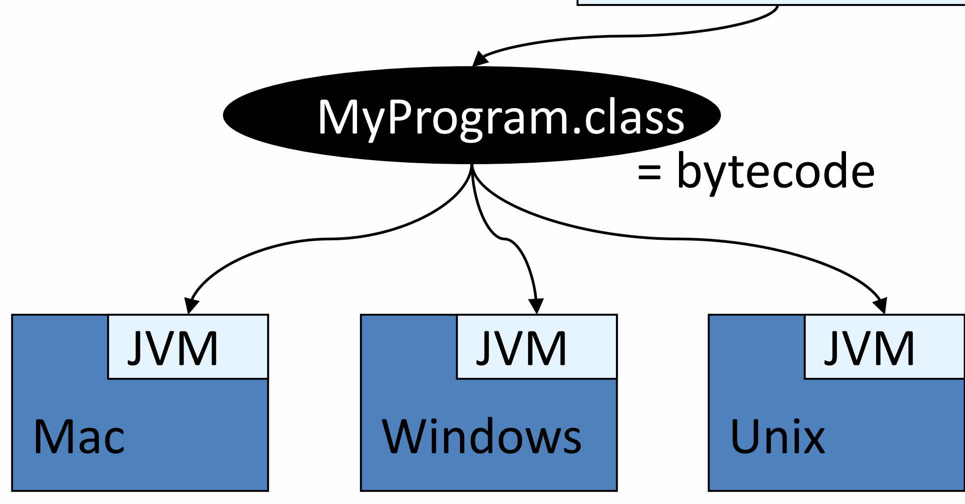
- Java est un langage de programmation simple
  - Langage de programmation au même titre que C/C++/Perl/Smalltalk/Fortran mais plus simple
  - Les aspects fondamentaux du langage sont rapidement assimilés
- Java est orienté objet :
  - La technologie OO après un moment de gestation est maintenant complètement intégrée
  - En java, tout est un objet (à la différence du C++ par ex.)
- Simple aussi parce qu'il comporte un grand nombre d'objets prédéfinis pour l'utilisateur
- Java est familier pour les programmeurs C++

Qualité et Java

- Conçu pour créer des logiciels hautement fiables
- Oblige le programmeur à garder à l’esprit les erreurs hardware et software
- Vérifications complètes à l’exécution et à la compilation
- Existence d’un « garbage collector » qui permet d’éviter les erreurs de gestion de la mémoire

Java ByteCode

- Il existe une grande diversité de systèmes d'exploitation
- Le compilateur Java génère un bytecode, c'est à dire un format intermédiaire, neutre architecturalement, conçu pour faire transiter efficacement le code vers des hardware différents et/ou plateformes différentes
- Le bytecode ne peut-être interprété que par le processeur de la JVM



Support

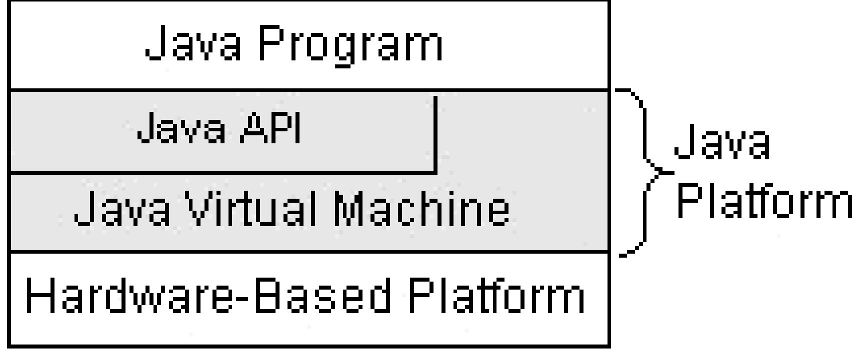
- Support intégré d'Internet
  - La Class URL
  - Communication réseaux TCP et UDP
  - RMI, CORBA, Servlets
  - JINI, pour les applications complexes distribuées…..
- Connectivité aux bases de données
  - JDBC: Java DataBase Connectivity
  - Offre des facilités de connexions à la plupart des BD du marché
  - Offre un pont vers ODBC
- Support des caractères internationaux
  - Java utilise le jeu de caractères UNICODE
  - JVM équipée de tables de conversion pour la plupart des caractères
  - JVM adapte automatiquement les paramètres régionaux en fonction de ceux de la machine sur laquelle elle tourne

Considération technique

- Considération basique
  - Est-ce que l'exécution ralentie à cause de l'interpréteur ? Java est équipe d'un JIT limitant la perte de performance
  - Le code natif généré par l'interpréteur est-il aussi rapide que celui réalisé par un compilateur classique (par ex C)? Non car c'est un langage différent.
- C'est un langage nativement MultiThread
  - Comparable au multitâche d'un OS
  - Le temps du CPU est divisé (sliced)
- L'édition de lien effectuée à l'exécution du programme
- Codes exécutables chargés depuis un serveur distant permet la mise à jour transparente des applications

Java comme Plateforme

- Plateforme = environnement hardware ou software sur lequel le programme est exécuté.
- La Java « Platform » se compose de:
  - la Java Virtual Machine (Java VM)
  - la Java Application Programming Interface (Java API)



API

- L'API Java est structuré en libraires (packages). Les packages comprennent des ensembles fonctionnels de composants (classes)..
  - On parle souvent de ClassPath
  - La librairie standard est appelé JRE pour Java Runtime Environnement.
- Le JRE comprend notamment :
  - Essentials (data types, objects, string, array, vector, I/O,date,…)
  - Applet
  - Abstract Windowing Toolkit (AWT)
  - Basic Networking (URL, Socket –TCP or UDP-,IP)
  - Evolved Networking (Remote Method Invocation)
  - Internationalization
  - Security
  - …..

La JVM

- « An imaginery machine that is implemented by emulating it in software on a real machine. Code for the JVM is stored in .class files, each of which contains code for at most one public class »
- Définit les spécifications hardware de la plateforme
- Lit le bytecode compilé (indépendant de la plateforme)
- Implémentée en software ou hardware
- Implémentée dans des environnements de développement ou dans les navigateurs Web

La JVM

La JVM définit :

- Les instructions de la CPU
- Les différents registres
- Le format des fichiers .class
- Le « stack »
- Le tas (« Heap ») des objets « garbage-collectés »
- L’espace mémoire

La JVM

Trois tâches principales :

- Charger le code (class loader)
- Vérifier le code (bytecode verifier)
- Exécuter le code (runtime interpreter)

D'autres THREAD s'exécutent :

- Garbage collector
- (JIT compiler)

Historique

- 1991: Développement de OAK
  - langage simple, portable et orienté objets
  - pour la programmation d'appareils électroniques ménagers
  - emprunte la portabilité du Pascal (VM) et la syntaxe de C++
- 1994: Abandon du projet OAK
  - Peu d'enthousiasme pour l'idée
- 1995: Intégration de la JVM dans Netscape
  - Apparition des Applets
  - Explosion d'Internet ➔ attrait grandissant pour Java
- 1999: Apparition de JINI
  - Nouvelle technologie basée sur Java
  - Reprend l'ambition de départ d'un plug and play universel
  - Distribué sur tous les appareils munis d'un processeur
- 2006: Java devient Open Source
  - Les sources de la plateformes Java sont désormais libres sous licence GNU

Version

- Java 1.4 (fév 2002) XML, JCE,
- J2SE 1.5(septembre 2004) Enumeration, Auto boxing, for each
- J2SE 1.6 (décembre 2006) Faille de sécurité
- J2SE 1.7 (juillet 2011), GPL, Future, multicatch
- J2SE 1.8 (mars 2014), fonction lambda, stream
- J2SE 1.9 (2015), JSON, HTTP2
- Java SE 10 (2018) Grall JIT
- Java SE 11 (2018) suppression de Corba, J2EE
- Java SE 12 (2019) GC G1, unification des JVM ARM 64

Introduction

Le framework Java EE, les différents serveurs Applicatif J2EE

- Qu’est ce que le framework J2EE au regard de Java SE
- Type de serveurs J2EE

Jakarta

- Java Enterprise Edition (Java EE), anciennement Java 2 Platform, Enterprise Edition (J2EE), actuellement renommé Jakarta EE, est un ensemble de spécifications qui étend Java SE 8 avec des spécifications pour des fonctionnalités d'entreprise telles que l'informatique répartie et les services Web.
- Les applications Java EE sont exécutées sur des environnements d'exécution de référence, qui peuvent être des microservices ou des serveurs d'applications, qui gèrent les transactions, la sécurité, l'évolutivité, la concurrence et la gestion des composants qu'il déploie.
- Java EE est défini par sa spécification. La spécification définit les API (interface de programmation d'application) et leurs interactions. Comme pour les autres spécifications Java Community Process, les fournisseurs doivent respecter certaines exigences de conformité pour pouvoir déclarer leurs produits conformes à Java EE.

Versions

- J2EE 1.2 (12 décembre 1999)
- J2EE 1.3 (24 septembre 2001)
- J2EE 1.4 (11 novembre 2003)
- Java EE 5 (11 mai 2006)
- Java EE 6 (10 décembre 2009)
- Java EE 7 (28 mai 2013 mais 5 avril 2013, conformément au document de spécification. Le 12 juin 2013 était la date de lancement prévue)
- Java EE 8 (31 août 2017)
- Jakarta EE 8 (10 septembre 2019) - entièrement compatible avec Java EE 8
- Java EE était géré par Oracle dans le cadre du processus de la communauté Java. Le 12 septembre 2017, Oracle Corporation a annoncé qu'elle soumettrait Java EE à la base Eclipse. [7] Le projet de niveau supérieur Eclipse a été nommé Eclipse Enterprise for Java (EE4J).La fondation Eclipse a été obligée de changer le nom de Java EE car Oracle est propriétaire de la marque du nom "Java". [9] Le 26 février 2018, il a été annoncé que le nouveau nom de Java EE serait Jakarta EE.

Architecture J2EE

Browser

html

http

WEB Container

Servlets
JSPs

EJBs

EJB Container

rmi

public
static void
main(…) {

Java Application

JDBC
JMS
JTA
JCA
JAAS
JavaMail
JNDI
…

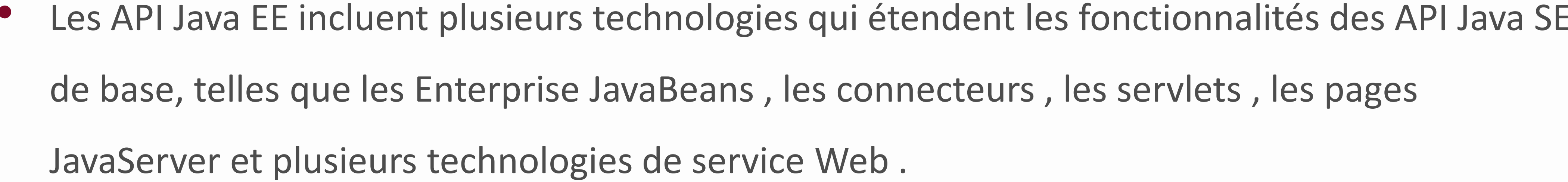
Java EE Application Server

DB

EIS

J2EE

- Java EE inclut plusieurs spécifications ayant différents objectifs, telles que la génération de pages Web, la lecture et l'écriture d'une base de données de manière transactionnelle, la gestion de files d'attente distribuées.



- Les API Java EE incluent plusieurs technologies qui étendent les fonctionnalités des API Java SE de base, telles que les Enterprise JavaBeans , les connecteurs , les servlets , les pages JavaServer et plusieurs technologies de service Web .

Spécifications Web

- Servlet : définit comment gérer les requêtes HTTP, de manière synchrone ou asynchrone. Il est de bas niveau et d'autres spécifications Java EE s'y reposent;
- WebSocket: la spécification Java API for WebSocket définit un ensemble d'API permettant de gérer les connexions WebSocket ;
- **Java Server Faces : une technologie permettant de créer des interfaces utilisateur à partir de composants**; Reposant sur une Servlet (Face Servlet)
- Unified Expression Language (EL) est un langage simple conçu à l'origine pour répondre aux besoins spécifiques des développeurs d'applications Web. Il est utilisé spécifiquement dans Java Server Faces pour lier des composants à des beans (sauvegarde) et dans Contexts and Dependency Injection pour nommer des beans, mais peut être utilisé sur l'ensemble de la plate-forme.

Spécifications du service Web

- **L'API Java pour les services Web RESTful prend en charge la création de services Web conformément au modèle d'architecture REST (Representational State Transfer);**
- L'API Java pour le traitement JSON est un ensemble de spécifications permettant de gérer les informations codées au format JSON.
- L'API Java pour JSON Binding fournit des spécifications permettant de convertir les informations JSON en ou à partir de classes Java.
- L'architecture Java pour la liaison XML permet de mapper le XML en objets Java;
- L'API Java pour les services Web XML peut être utilisée pour créer des services Web SOAP

Spécifications d'entreprise

- **Contexts and Dependency Injection est une spécification visant à fournir un conteneur d'injection de dépendance, comme dans Spring;**
- **La spécification Enterprise JavaBean (EJB) définit un ensemble d'API allégées qu'un conteneur d'objets (le conteneur EJB) prendra en charge**
- **Les API Java Persistence sont des spécifications relatives au mappage objet-relationnel entre les tables de bases de données relationnelles et les classes Java.**
- L'API Java Transaction contient les interfaces et les annotations permettant d'interagir avec le support des transactions offert par Java EE. Même si cette API résume les détails de très bas niveau, les interfaces sont également considérées comme de bas niveau et le développeur d'applications moyen dans Java EE est supposé soit s'appuyer sur le traitement transparent des transactions par les abstractions d'EJB de niveau supérieur, soit en utilisant les annotations fournies par cette API en combinaison avec les beans gérés par CDI.
- Java Message Service offre aux programmes Java un moyen courant de créer, d'envoyer, de recevoir et de lire les messages d'un système de messagerie d'entreprise.

Autres spécifications

- **Validation**: ce package contient les annotations et les interfaces pour le support de validation déclarative offert par l'API de validation de bean. Dans Java EE, JPA respecte les contraintes de validation des beans dans la couche de persistance, alors que JSF le fait dans la couche de vue.
- **Les Batch API** permettent d'exécuter de longues tâches d'arrière-plan impliquant éventuellement un volume de données important et devant être exécutées périodiquement.
- Java EE Connector Architecture est un outil Java permettant de connecter des serveurs d'applications et des systèmes d'information d'entreprise (EIS) dans le cadre de l'intégration d'applications d'entreprise (EAI). Il s'agit d'une API de bas niveau destinée aux fournisseurs avec lesquels le développeur d'applications moyen ne communique généralement pas.

Quelques serveurs J2EE

- GlassFish (Oracle) : Implémentation de référence
- Wildfly (Redhat) : version Open Source de Jboss EAP
- **JBoss EAP (Redhat)**
- IBM WebSphere AS

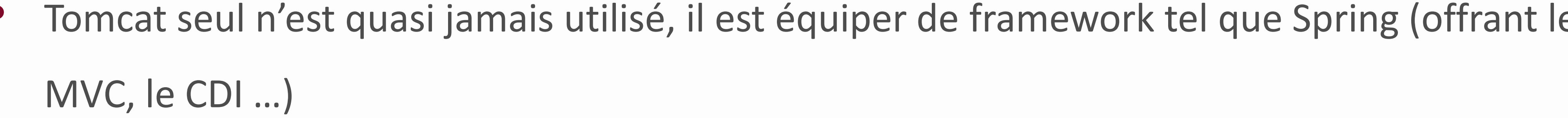
JBoss vs Tomcat

- Dans Java Enterprise Edition, les serveurs d'applications peuvent également être divisés logiquement en un conteneur de servlet, un conteneur de client d'application et un conteneur d'EJB..
- **Conteneur client d'application** fournit l'injection de dépendance et la sécurité.
- **Conteneur EJB** peut exécuter le cycle de vie des EJB et est capable de gérer des transactions
- Le serveur d'applications JBoss fournit une pile Java Enterprise Edition (Java EE) complète, y compris Enterprise Java Beans (EJB) et de nombreuses autres technologies..

JBoss vs Tomcat

- Tomcat est un serveur Web open source et un conteneur de servlets. Apache Software Foundation l'a développé. Il peut exécuter des servlets et JSP (Java Server Pages).
- Il fournit un environnement de serveur Web Java pur pour exécuter des applications Java. Apache Tomcat comprend des outils de configuration et de gestion. Les configurations directes peuvent être effectuées en modifiant les fichiers de configuration XML..

JBoss vs Spring



- Tomcat seul n’est quasi jamais utilisé, il est équiper de framework tel que Spring (offrant le MVC, le CDI …)
- Le mythe le plus répandu est que les serveurs d'applications Java EE sont lourds et que Spring est une plate-forme plus légère.
  Cela n’a plus aucun sens: en fait, les serveurs d’applications sont également extrêmement rapides et légers.
- En fait la différence est que J2EE est normé et Spring est plus libre (moins d’acteur). Spring innove donc relativement plus rapidement sans garantie sur la pérennité des composants

Introduction

Compilation et exécution du programme.

- Contenu
  - Les outils Java/Javac
  - Créer un programme
- Objectif:
  - Ecrire un programme simple
  - Le compiler
  - L’executer

Java/Javac

- Le langage « Java » est un langage qui se compile avec l'utilitaire « Javac » vers une machine virtuelle « Java ».
- Le compilateur s'appel javac et la machine java

- jdk1.7.0_51
  - bin
  - db
    - bin
    - lib
  - include
  - jre
    - bin
    - lib
  - lib
  - src.zip
- jre7

Nom	Modifié le	Type	Taille
appletviewer.exe	15/02/2014 19:59	Application	16 Ko
apt.exe	15/02/2014 19:59	Application	16 Ko
extcheck.exe	15/02/2014 19:59	Application	16 Ko
idlj.exe	15/02/2014 19:59	Application	16 Ko
jabswitch.exe	15/02/2014 19:59	Application	55 Ko
jar.exe	15/02/2014 19:59	Application	16 Ko
jarsigner.exe	15/02/2014 19:59	Application	16 Ko
java.exe	15/02/2014 19:59	Application	185 Ko
javac.exe	15/02/2014 19:59	Application	16 Ko
javadoc.exe	15/02/2014 19:59	Application	16 Ko
javafxpackager.exe	15/02/2014 19:59	Application	79 Ko
javah.exe	15/02/2014 19:59	Application	16 Ko
javap.exe	15/02/2014 19:59	Application	16 Ko
java-rmi.exe	15/02/2014 19:59	Application	16 Ko
javaw.exe	15/02/2014 19:59	Application	185 Ko
javaws.exe	15/02/2014 19:59	Application	306 Ko
jcmd.exe	15/02/2014 19:59	Application	16 Ko
jconsole.exe	15/02/2014 19:59	Application	17 Ko
jdb.exe	15/02/2014 19:59	Application	16 Ko
jhat.exe	15/02/2014 19:59	Application	16 Ko
jinfo.exe	15/02/2014 19:59	Application	16 Ko
jli.dll	15/02/2014 19:59	Extension de l'app...	154 Ko

Java/Javac

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

```
C:\Users\pilou\>javac.exe test.java
C:\Users\pilou\>dir
15/04/201412:56 408 test.class
15/04/201412:56 102 test.java
C:\Users\pilou\>java.exe test
Coucou
```

Java/Javac

```
C:\Users\pilou\>javap.exe -c test
Compiled from "test.java"
public class test {
public test();
Code:
 0: aload_0
 1: invokespecial #1// Method java/lang/Object."<init>":()V
 4: return

public static void main(java.lang.String[]);
Code:
 0: getstatic #2// Field java/lang/System.out:Ljava/io/PrintStream;
 3: ldc #3// String Coucou
 5: invokevirtual #4// Method java/io/PrintStream.println:(Ljava/lang/String;)V
 8: return
}
```

Genèse d'un premier programme

Ecriture d’un programme simple.

- Contenu
  - Voir la syntaxe
- Objectif:
  - Comprendre un programme simple

Java

```
/**
On définit une classe appelé test ayant une méthode « static » noté main
Prennant un tableau de chaîne de caractère (String []) noté args
*/
public class MonPremierProgramme
{
public static void main(String [] args)
{
               /* on demande la classe System
               Ayant un attribut static noté out
               Cet attribut est un objet ayant une méthode
               println
               */
               System.out.println("Coucou");
}
}



C:\java MonPremierProgramme
>Coucou
```

Fonction Main

- La fonction **main** est la fonction principale des programmes en Java: Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction **main**.
- Définition de la fonction main « public static void main(String [] args)
- Remarque avancée:
  - Il est possible de faire passer des arguments de la ligne de commande à un programme.

Genèse d'un premier programme

Qu’est ce qu’une librairie

- Contenu
  - Une librairie
  - Les fichiers « jar »
- Objectif:
  - Comprendre ce qu’est une librairie
  - Entrevoir le monde du JRE

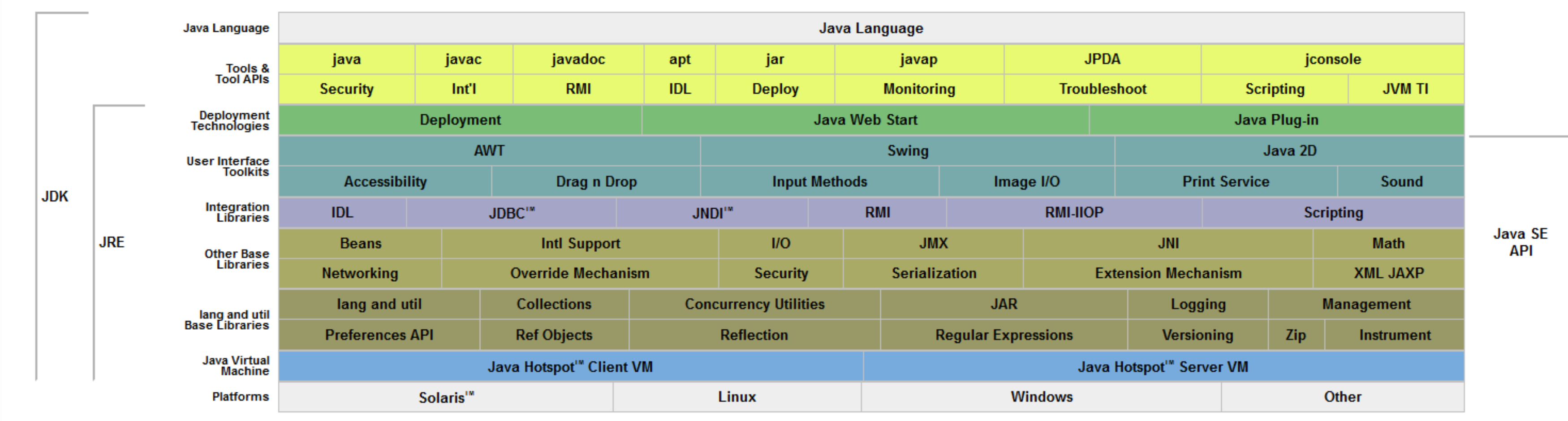
Une librairie

- Un programme Java ou autre est rarement écrit à 100%
- Il s'agit en fait de l'adjonction de composant « pré-écrit » et de1 à 5% de code « maison ».
- L'idée est qu'un composant pré-écrit est a vécu l'épreuve du temps:
    - Plus robuste (moins d'erreur)
    - Plus facile d'utilisation
    - Plus d'évolutivité

Des librairies

- Il existe 1 librairie particulière qui est le celle du JRE (Java Runtime Environnement) qui est incluse par défaut dans tous les programmes Java.
- Sinon il existe des librairies pour faire de la 3D, de l'image du son, du XML ……
- Lors de l'utilisation d'une librairie X, on parle de l'importation de cette librairie.

Les librairies standard de Java



Utilisation des librairies en Java

- Java.lang : la librairie de « base » de Java avec la classe « System »
- Les autres librairies « standard » s'utilise avec la directive import. Par exemple, pour une liste d'objet on écrie import java.util.ArrayList
- Les autres librairies doivent en plus être précisé à l'execution
- La collection des librairies utilisé par un programme est vulgairement appelé « classpath ».

Introduction

Eclipse

- Prise en main d’Eclipse
- Visualisation des principales fonctions

L’environnement Eclipse

- Eclipse est un Environnement de Développement Intégré (IDE)
- Spécialement conçu pour le développement en Java
- Créé à l’origine par IBM
- Puis cédé à la communauté Open Source
- Caractéristiques principales
  - Notion de « projet » (1 programme ➔ 1 projet)
  - Colore le code en fonction de la signification des mots utilisés
  - Force l’indentation du code
  - Compile le code en temps réel
    ➔ Identifie les erreurs en cours de frappe
  - Peut générer des bouts de code automatiquement
  - Permet de gérer le lancement des applications

Créer un projet Eclipse – Etape 1

Java - Eclipse SDK

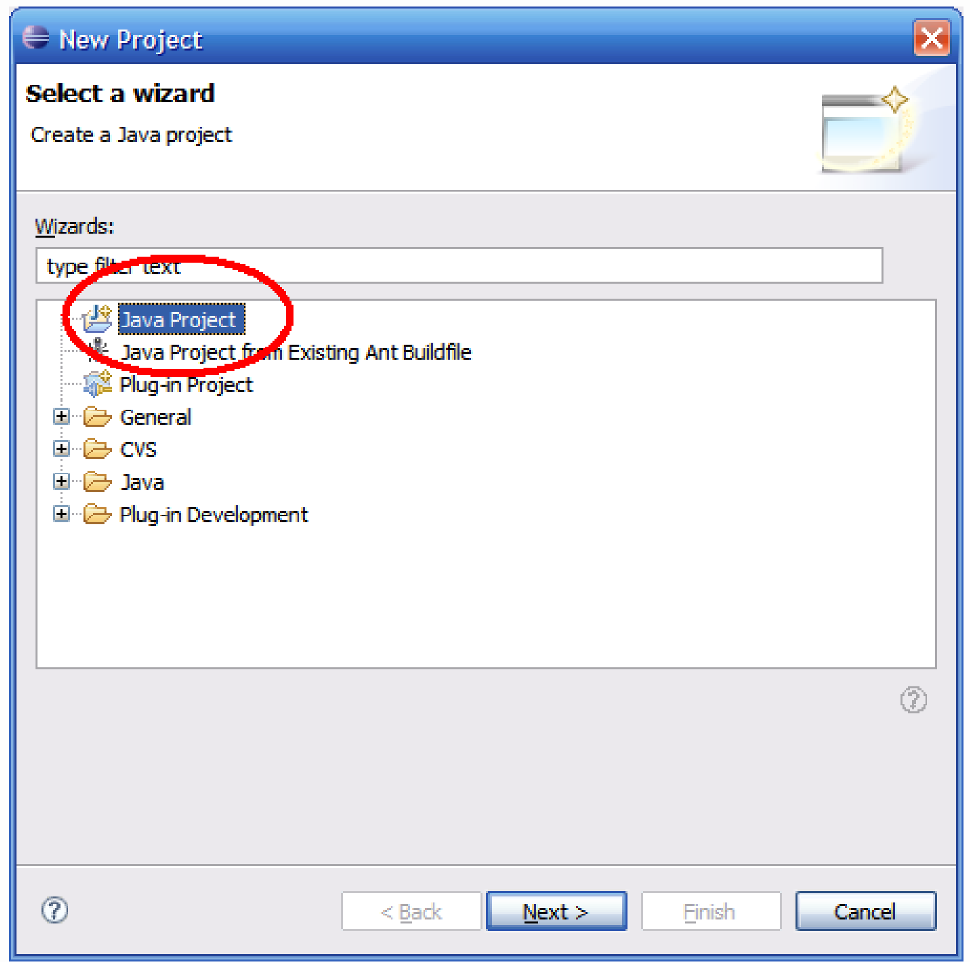
File Edit Source Refactor Navigate Search Project Run

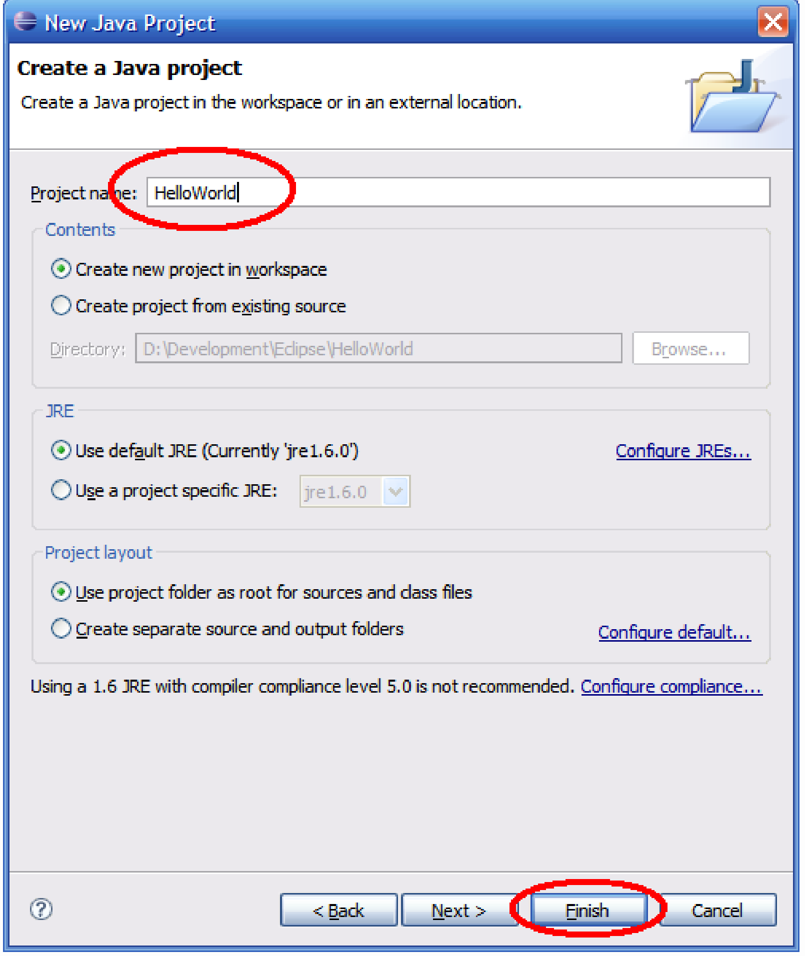
File menu	Shortcut
New	Alt+Shift+
Open File…	
Close	Ctrl+W
Close All	Ctrl+Shift+W
Save	Ctrl+S
Save As…	
Save All	Ctrl+Shift+S
Revert	
Move…	
Rename…	F2
Refresh	F5
Convert Line Delimiters To	
Print…	Ctrl+P
Switch Workspace…	
Import…	
Export…	
Properties	Alt+Enter
1 ReservationGroupe.java [Solvay2006/…]	
2 ReservationIndividuelle.java [Solva…]	
3 Reservation.java [Solvay2006/…]	
4 Representation.java [Solvay2006/…]	
Exit	

New submenu:

- Project…
- Package
- Class
- Interface
- Enum
- Annotation
- Source Folder
- Folder
- File
- Untitled Text File
- JUnit Test Case
- Other…

Créer un projet Eclipse – Etape 2





Créer un projet Eclipse – Etape 3

Java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package ... Hierarchy Navigator

- Banque
- HelloWorld
  - JRE System Library [jre1.6.0]
- Paint
- Solvay2006
- TestDBPool
- TestDBSwing

Le contenu de votre projet
(les classes) apparaîtra ici

Une première application en Java

- Maintenant que notre projet a été créé, nous pouvons commencer à développer une application
- Une application Java est composée de « Classes »
  - En règle générale, chaque classe correspond à un fichier
  - Chaque fichier « source » (le code de chaque classe) est sauvé avec un nom de fichier correspondant au nom de la classe et l'extension « .java »
  - Java est dit « case-sensitive » ➔ Distingue majuscules et minuscules!!!
- Notre première application sera composée d'une seule classe
  - Le nom de cette classe sera « HelloWorld »
  - Elle sera donc enregistrée dans un fichier nommé « HelloWorld.java »
  - Le code de cette classe (fourni plus loin) doit être recopié tel quel
  - <u>**ATTENTION**</u>
    - Chaque symbole importe
    - Une majuscule n'est pas une minuscule

Une première application en Java

Java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace...
Import...
Export...
Properties Alt+Enter
1 ReservationGroupe.java [Solvay2006/...]
2 ReservationIndividuelle.java [Solva...]
3 Reservation.java [Solvay2006/...]
4 Representation.java [Solvay2006/...]
Exit

Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Folder
File
Untitled Text File
JUnit Test Case
Other...

New Java Class

Java Class

The use of the default package is discouraged.

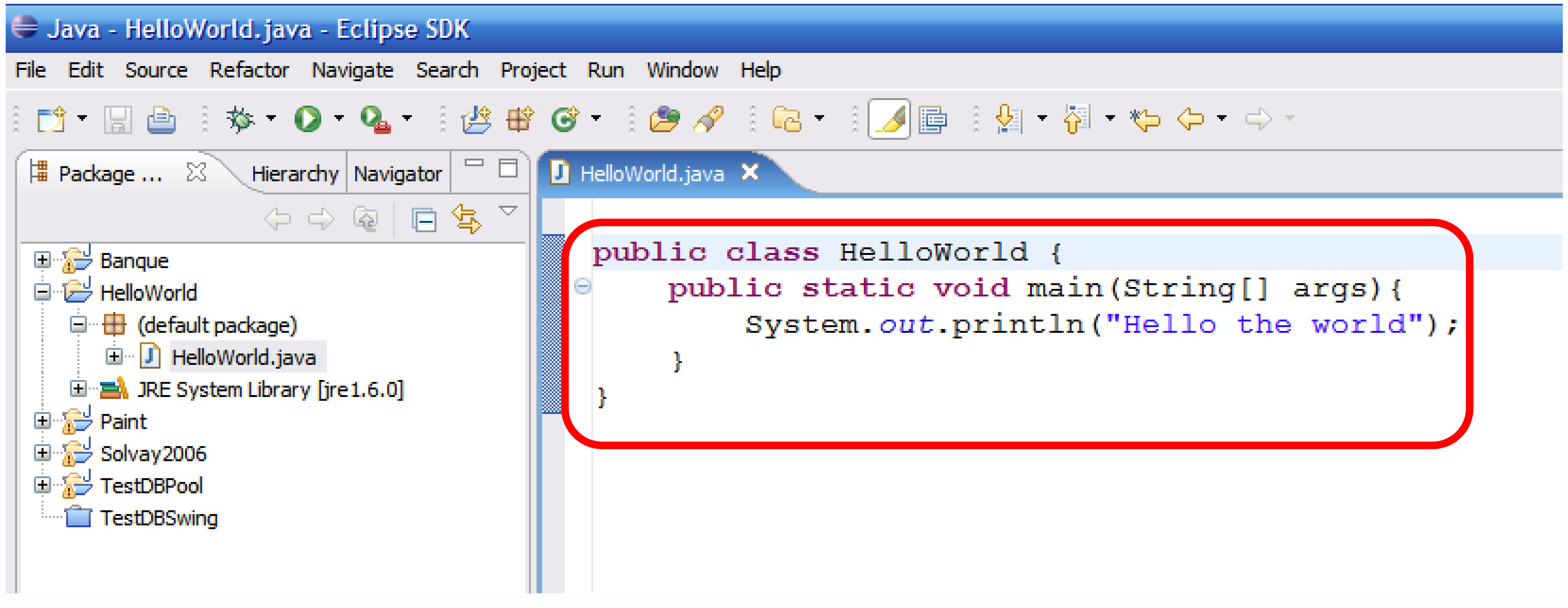
Source folder: HelloWorld Browse...
Package: (default) Browse...
Enclosing type: Browse...
Name: HelloWorld
Modifiers: public default private protected
abstract final static
Superclass: java.lang.Object Browse...
Interfaces: Add... Remove

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 as configured in the properties of the current project?
Generate comments

Finish Cancel

Une première application en Java




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

La première ligne du programme doit être la déclaration de la classe

Tout programme doit contenir une méthode **main** qui porte la signature ci-contre

Écrire à l'écran "Hello the World"

Fermer les accolades

Une première application en Java

- Une fois le programme écrit (ici l'unique classe), il reste à le lancer
- Pour lancer une application Java, il faut
  - La compiler (***fait automatiquement par Eclipse***)
  - Lancer la machine virtuelle (JVM) (***fait automatiquement par Eclipse***)
  - Ordonner à la JVM d'appeler la méthode « main » de la classe principale

    ➔ Créer une « configuration de lancement » dans Eclipse

    ➔ Pour « apprendre » à Eclipse comment lancer notre programme

    ➔ Une fois cette configuration créée, on pourra relancer le programme en cliquant simplement sur le bouton

Une première application en Java
Créer une configuration de lancement Eclipse

Java - HelloWorld.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package ... Hier

- Banque
- HelloWorld
  - (default package
    - HelloWorld.j
  - JRE System Libra
- Paint
- Solvay2006
- TestDBPool
- TestDBSwing

1 FenetrePrincipale
2 PaintApplet
3 LancePaint
4 LanceBanque
Run As
Run...
Organize Favorites...

HelloWorld.java

```
public class HelloWorl
    public static void
        System.out.pri
    }
}
```

Une première application en Java
Créer une configuration de lancement Eclipse

Run

Create, manage, and run configurations

Run a Java application

2

type filter text

- Eclipse Application
- Equinox OSGi Framework
- Java Applet
  - PaintApplet
- Java Application
- JUnit
- JUnit Plug-in Test
- SWT Application

1

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Run | Close

Une première application en Java
Créer une configuration de lancement Eclipse

Run

Create, manage, and run configurations

Run a Java application

type filter text

- Eclipse Application
- Equinox OSGi Framework
- Java Applet
  - PaintApplet
- Java Application
  - FenetrePrincipale
  - HelloWorld
  - LanceBanque
  - LancePaint
  - New_configuration
- JUnit
- JUnit Plug-in Test
- SWT Application

Name: HelloWorld

Main | (x)= Arguments

Project:
HelloWorld | Browse...

Main class:
HelloWorld | Search...

- [ ] Include libraries when searching for a main class
- [ ] Include inherited mains whe
- [ ] Stop in main

Apply | Revert

Run | Close

1 Nom de la configuration de lancement

2 Nom de votre projet (indiqué automatiqueme

3 Nom de la classe principale
(normalement indiqué automatiquement)

4 Cliquez sur « Run » pour lancer le programme

Une première application en Java
Le résultat de votre application

Java - HelloWorld.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Cliquez sur le bouton « Run » pour relancer votre programme

Package ... Hierarchy Navigator

- Banque
- HelloWorld
  - (default package)
    - HelloWorld.java
  - JRE System Library [jre1.6.0]
- Paint
- Solvay2006
- TestDBPool
- TestDBSwing

HelloWorld.java

```java
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("Hello the world");
    }
}
```

La « Console » affiche le résultat de votre programme

Problems Javadoc Declaration Console

<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe (2 déc. 06 09:16:20)

```
Hello the world
```

Writable Smart Insert 7 : 1

Introduction

Debuggage

- Voir la vue de debuggage sous Eclipse

Le bouton debuggage

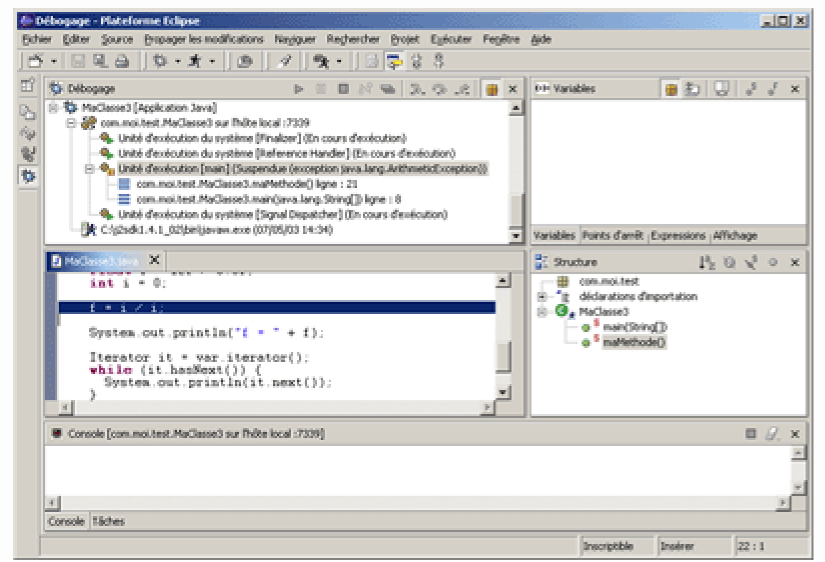
- Pour déboguer du code Java, Eclipse propose une perspective dédiée : la perspective "Débogage".

- Celle ci est automatiquement affichée lorsqu'une application est lancée sous le contrôle du débogueur en utilisant le bouton de la barre d'outils. Son principe de fonctionnement est identique au bouton d'exécution situé juste à côté de lui.

Vue debuggage

Par défaut, la perspective "Débogage" affiche quelques vues aussi présentes dans la perspective Java (les vues "Structure" et "Console") ainsi que l'éditeur de code Java.

Elle affiche aussi plusieurs vues particulièrement dédiées au débogage.



Les vues de debugage

Les vues spécifiques au débogage sont :

- la vue "Débogage" : affiche sous la forme d'une arborescence, les différents processus en cours d'exécution ou terminés
- la vue "Variables" : affiche les variables utilisées dans les traitements en cours de débogage
- la vue "Points d'arrêts" : affiche la liste des points d'arrêts définis dans l'espace de travail
- la vue "Expressions" : permet d'inspecter une expression en fonction du contexte des données en cours d'exécution
- la vue "Affichage" : permet d'afficher le résultat de l'évaluation d'une expression

la vue "Débogage"

- Se déclenche lorsque le programme s’arrête (sur un point d’arrêt)
- une exception non capturée est propagée jusqu'au sommet de la pile d'appel

▶	Reprendre l'exécution précédemment interrompue
⏸	Interrompre l'exécution du processus
■	Demande de mettre fin au processu
	Exécute la ligne courante et arrête l'exécution sur la première ligne de code incluse dans la première méthode de la ligne courante
	Exécute la ligne courante et arrête l'exécution avant la ligne suivante
	Exécute le code de la ligne courante jusqu'à la prochaine instruction return de la méthode

La vue « Points d'arrêts »

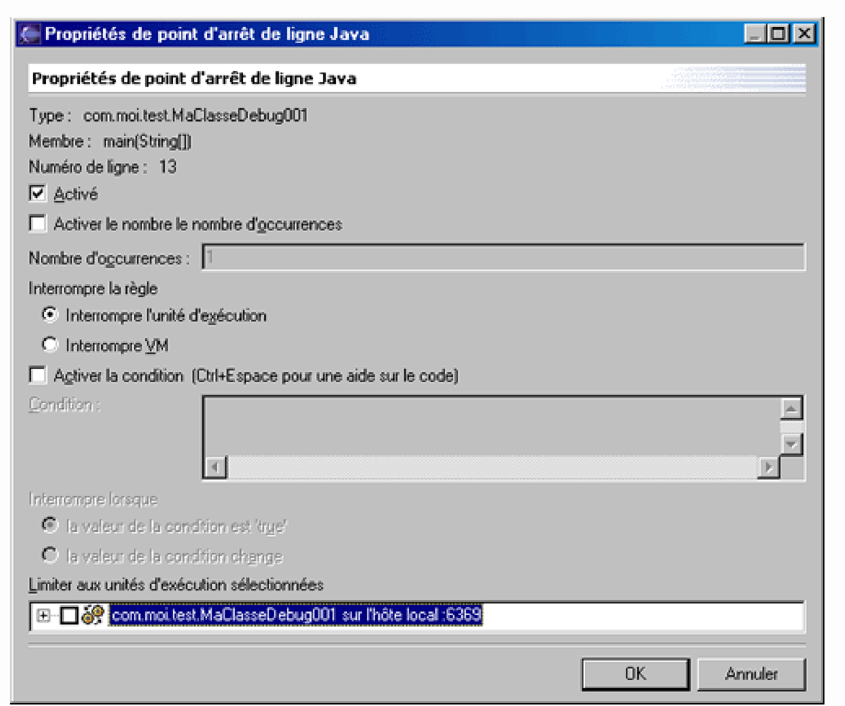
Pour placer un point d'arrêt, il suffit dans l'éditeur de double cliquer dans la barre de gauche pour faire apparaître une icône ronde bleue.

MaClasse.java

```
package com.moi.test;

public class MaClasse {

    public static void main(String[] args) {
        int valeur;
        valeur = 10;
        System.out.println("valeur = "+valeur);
    }

}
```

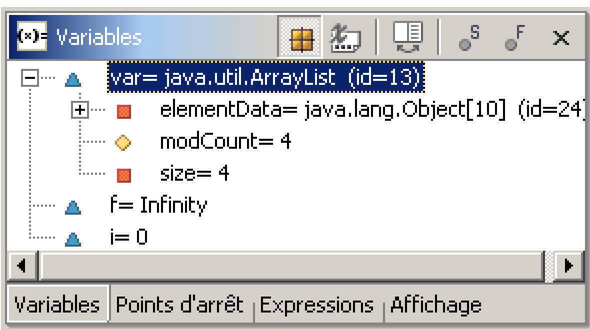
La vue « Points d'arrêts »

- Un click droit sur une ligne de code permet de positionner un point d’arret.
- Il est possible d'activer ou de désactiver le point d'arrêt sélectionné respectivement grace à l'option "Activer" ou "Désactiver".
- Un de ces paramètres les plus intéressants est la possibilité de mettre une condition d'activation du point d'arrêt. Il suffit pour cela de cocher la case "Activer la condition" et de la saisir dans la zone de texte prévue à cet effet. Dans cette zone de texte, l'assistant de complétion de code est utilisable.



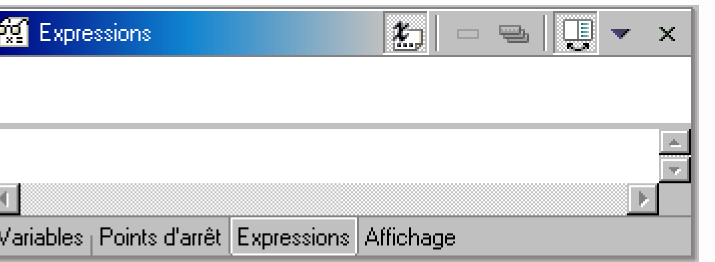
La vue « variables »

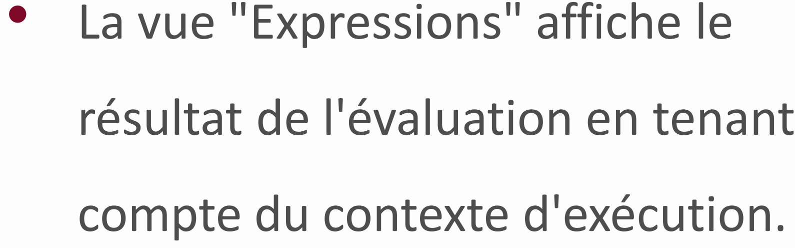
- Cette vue permet de visualiser les valeurs des variables utilisées dans les traitements en cours de débogage. Ces valeurs peuvent être unique si la variable est de type primitive ou former une arborescence contenant chacun des champs si la variable est un objet.



La vue expression

- La vue "Expressions" permet d'inspecter la valeur d'une expression selon les valeurs des variables dans les traitements en cours de débogage.





- La vue "Expressions" affiche le résultat de l'évaluation en tenant compte du contexte d'exécution.

Introduction à la programmation avec Java

- Introduction
- Les constructions de base d'un programmes
  - Les variables : déclaration et typage.
  - Les méthodes : définition.
  - Les expressions.
  - Les instructions de contrôle : les instructions conditionnelles, de boucle, de branchement.
  - Les tableaux.
  - Les unités de compilation et packages : le contrôle de la visibilité des classes, le mécanisme d'import.
  - Les imports statiques.
- Tests Logiciels
- Bonne Pratiques de conception d'une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d'un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Règles de programmation

Convention de nommage/syntaxe.

- Contenu:
  - Comprendre l’intérêt d’une convention de nommage
  - Connaitre les conventions de nommage
- Objectif:
  - Connaitre les conventions de Java

Convention de nommage

Chaque objet, classe, programme ou variable est associé à un nom : l'identificateur qui peut se composer de tous les caractères alphanumériques et des caractères _ et $. Le premier caractère doit être une lettre, le caractère de soulignement ou le signe dollar.

Rappel : Java est sensible à la casse.

Un identificateur ne peut pas appartenir à la liste des mots réservés du langage Java :

abstract	const	final	int	public	throw
assert (Java 1.4)	continue	finally	interface	return	throws
boolean	default	float	long	short	transient
break	do	for	native	static	true
byte	double	goto	new	strictfp	try
case	else	if	null	super	void
catch	enum (Java 5)	implements	package	switch	volatile
char	extends	import	private	synchronized	while
class	false	instanceof	protected	this	

Convention de nommage

- Une convention de nommage est un ensemble de règle « commune » pour l’écriture de programme.
- Le respect de ces règles ne sont pas une obligation du langage Java mais une preuve de politesse vis-à-vis des autres développeurs.
- De plus, ces règles ne sont pas là par « hasard ». Elles améliorent la lisibilité et donc diminue les problèmes de programmation.

Convention classique

- Le camelCase c'est la manière la plus utilisée pour nommer en Java.
  - Chaque première lettre d'un mot prend une majuscule
  - tous les mots sont collés les uns aux autres
  - le premier mot ne prend pas de majuscule.

- Un exemple de camelCase est : ceciEstUnExemple.

Convention classique

- Toutes les variables sont en camelCase
- Les noms de classes sont en camelCase avec première lettre en majuscule.
- Les noms de méthodes sont en camelCase.
- Les constantes sont en majuscule
- Les variables i,j,k,l sont souvent associé aux entiers et c,d,e souvent associé au charactere.

Convention classique

```
class Example {
int[] myArray = { 1, 2, 3, 4, 5, 6 };
int theInt = 1;
String someString = "Hello";
double aDouble = 3.0;

void foo(int a, int b, int c, int d, int e, int f) {
         switch (a) {
         case 0:
         Other.doFoo();
         break;
         default:
         Other.doBaz();
         }
}

void bar(List v) {
         for (int i = 0; i < 10; i++) {
         v.add(new Integer(i));
         }
}
}
```

Convention classique

- Le symbole { définit une entrée de bloc et est toujours sur la même ligne que la déclaration précédente
- Le symbole } fin de de bloc est toujours précédé d'un retour chariot
- Après chaque { on rajoute une indentation afin de bien voir le bloc. En Java une indentation est soit une tabulation soit 4 espaces.
- En pratique on laisse Eclipse mettre en place la convention de syntaxe

Règles de programmation

Utilisation des commentaires. Pourquoi commenter les développements ?

- Contenu:
  - Qu’est ce qu’un commentaire?
  - Ou positionner les commentaires
- Objectif:
  - Comprendre pourquoi il faut commenter?
  - Comprendre le taux de commentaires.

Les commentaires

- Ils ne sont pas pris en compte par le compilateur donc ils ne sont pas inclus dans le pseudo code. Ils ne se terminent pas par un caractère ";".
- Il existe trois types de commentaire en Java :

Type de commentaires	Exemple
commentaire abrégé	// commentaire sur une seule ligne int N=1; // déclaration du compteur
commentaire multi ligne	/* commentaires ligne 1 commentaires ligne 2 */
commentaire de documentation automatique	/** * commentaire de la methode * @param val la valeur à traiter * @since 1.0 * @return la valeur de retour * @deprecated Utiliser la nouvelle methode XXX */

Commentaires

- Les commentaires sont des textes au format libre aidant à donner du sens a un programme informatique.
- L’idée est de faciliter la relecture, la maintenance et le partage d’information.
- NB: le premier des commentaires est le code source lui-même qui se doit d’être lisible

Quelques soit le commentaire …

```
#include "stdio.h"
#define xyxx char
#define xyyxx putchar
#define xyyyxx while
#define xxyyyx int
#define xxxyyx main
#define xyxyxy if
#define xyyxyy '\n'
xyxx *xyx [] = {
"]I^x[I]k\\I^o[IZ~\\IZ~[I^|[I^l[I^j[I^}[I^n[I]m\\I]h",
"]IZx\\IZx[IZk\\IZk[IZo_IZ~\\IZ~[IZ|_IZl_IZj\\IZj]IZ}]IZn_IZm\\IZm_IZh
",
"]IZx\\IZx[I^k[I\\o]IZ~\\IZ~\\I]|[IZl_I^j]IZ}]I^n[IZm\\IZm_IZh",
"]IZx\\IZx[IZk\\IZk[IZo_IZ~\\IZ~_IZ|[IZl_IZj\\IZj]IZ}]IZn_IZm\\IZm]IZh
",
"]I^x[I]k\\IZo_I^~[I^|[I^l[IZj\\IZj]IZ}]I^n[I]m^IZh",'\0'};/*xyyxyxyxx
xyxxxyy*/
xyxx *xyyx; xxyyyx
xyyyx,xyyyyx,xyyyyyx=0x59,xyyyyyyx=0x29,/*yxxyxyyyxxyyyxyy*/
xxyx=0x68;xxxyyx(){xyyyyx=0;xyyyxx(xyx[xyyyyx]){xyyx=xyx[xyyyyx++];/*x
yyyxxyx*/
xyyyxx(*xyyx){xyyyx= *xyyx++-xyyyyyx;xyyyxx(xyyyx--)xyyxx(*xyyx-
xyyyyyyx);/*x*/
xyxyxy(*xyyx==xxyx)xyyxx(xyyxyy);*xyyx++;}}}/*xyxyxyyyyxxyxxxyyyxyyyxy
xxyyy*/
```

Quoi/Comment commenté

- L'idée est de commenter :
  - Tous les points d'entrée d'une classe (attribut, méthode..)
  - Les parties de code « difficile » (algorithme)
  - Les choix d'implémentations
- Les commentaires doivent être instructif et concie.
- Il ne doivent pas dépasser en nombre de lignes le nombre de lignes de code.
- En Java, les classes, les méthodes public et protégé, les packages

Les variables

Qu’est ce qu’une variables

- Contenu
  - Description d’une variable
- Objectif:
  - Comprendre ce qu’est une variable
  - Voir la différence entre une variable de méthode et une variable de classe

Les variables

- Un programme Java ou autre utilise de la mémoire de l'ordinateur.
- La mémoire de l'ordinateur est vue par Java comme étant un ensemble de boite portant un nom et pouvant contenir un objet (de type chaussure, balle ….).
- On parle du :
  - nom de la variable pour le nom de la boite
  - Du type de la variable pour le type de la boite

Les variables

- Une variable possède un nom, un type et une valeur. La déclaration d'une variable doit donc contenir deux choses : un nom et le type de données qu'elle peut contenir. Une variable est utilisable dans le bloc où elle est définie.
- La déclaration d'une variable permet de réserver la mémoire pour en stocker la valeur.
- Le type d'une variable peut être :
  - soit un type élémentaire dit aussi type primitif déclaré sous la forme type_élémentaire variable;
  - soit une classe déclarée sous la forme classe variable ;

```
long nombre;
int compteur;
String chaine;
```

Les variables

- Java est un langage à typage rigoureux qui ne possède pas de transtypage automatique lorsque ce transtypage risque de conduire à une perte d'information.
- Pour les objets, il est nécessaire en plus de la déclaration de la variable de créer un objet avant de pouvoir l'utiliser. Il faut réserver de la mémoire pour la création d'un objet (remarque : un tableau est un objet en Java) avec l'instruction new. La libération de la mémoire se fait automatiquement grâce au garbage collector.
- Il est possible en une seule instruction de faire la déclaration et l'affectation d'une valeur à une variable ou plusieurs variables.

```
MaClasse instance; // déclaration de l'objet


instance = new MaClasse(); // création de l'objet


MaClasse instance = new MaClasse(); // déclaration //et création de
l'objet
```

Les variables

Les types primitifs : entiers, chaînes de caractères, nombres réels, autres.

- Contenu
  - Liste des types de variables
- Objectif:
  - Connaitre quelques « grand » types de variables

Les types de variables

- Une variable se définit comme une boite où l’on va mettre des « objets ».
- Une boite de banane ne peut contenir qu’une banane et une boite d’ananas contient des ananas
- Dire qu’une boite de X ne peut contenir que des X c’est donner la définition du type d’une variable.
- Une boite de X se dit en informatique une variable de type X

Les principaux types primitifs

- Parmis les types de variables :
- Les « booléen » qui peut contenir la valeur « vraie » ou la valeur « fausse »
- Les entiers ou integer qui contiennent des nombres entier (1,2,3 …)
- Les nombres floattant qui contiennent des nombres avec des virgules (1.2, 0.002 …)
- Le type « rien » ou « void » utiliser dans le cadre des fonctions/procedure.

Les principaux types primitifs

- Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution : c'est un des éléments qui permet à Java d'être indépendant de la plate-forme sur laquelle le code s'exécute.

- Les types élémentaires commencent tous par une minuscule.

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique : true ou false	1 bit	true ou false	pas de conversion possible vers un autre type
byte	octet signé	8 bits	-128 à 127	
short	entier court signé	16 bits	-32768 à 32767	
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans du code Java
int	entier signé	32 bits	-2147483648 à 2147483647	
float	virgule flottante simple précision (IEEE754)	32 bits	1.401e-045 à 3.40282e+038	
double	virgule flottante double précision (IEEE754)	64 bits	2.22507e-308 à 1.79769e+308	
long	entier long	64 bits	-9223372036854775808 à 9223372036854775807	

Les types « objets »

- Le type « objet » est un type définie par l'utilisateur ayant un nom choisie par l'utilisateur.
- Les types « objets » sont des assemblages de boites de type différents.
- Par Exemple, un point dans l'espace est composé de coordonnées. Soit une boite entière pour les X et une boites entière pour les Y.
- Un type objet porte un nom avec une majuscules.

Les types tableaux

- Il peut être utile de définir qu’une boite peut contenir d’autre boite.
- Par exemple, on veut pouvoir définir une boite spécial qui contiendras d’autre boite de bananes.
- C’est la notion du type tableau qui est un type spécial pouvant contenir des boites.

Les variables

Déclaration, définition et initialisation d'une variable.

- Contenu:
  - Commencer l'usage des variables
- Objectifs:
  - Déclarer une variables
  - Mettre une valeur dans une variables
  - Initialiser une variable
  - Comprendre la valeur null

Déclaration, définition et initialisation d'une variable.

```
Class Test
{
Public static void main(String [] args)
{
// on définit une variable par
// type de variable<espace> nom de variable
}
}
```

Déclaration d’un variable

- un nom de variable ne peut comporter que des lettres, des chiffres (les caractères _ et $ peuvent être utilisés mais ne devrait pas l'être pour des variables)
- Un nom de variable ne peut commencer par un chiffre et comporter d’espace
- Les noms de variables ne peuvent pas être les noms réservés du langage

Déclaration d'une variable.

```
Class Test
{
Public static void main(String [] args)
{
// déclaration d'un variable b de type booléen
boolean b;
// de même pour un entier du nom i
int i;
// ou pour un nombre a virgule
float f;

}
}
```

Définition d'une variable.

```
Class Test
{
Public static void main(String [] args)
{
// déclaration d'un variable b de type booléen
Boolean b;
// définition de sa valeur
b = true;
// de même pour un entier du nom i
Int i;
i=2;
// ou pour un nombre a virgule
Float f;
f=3.3;
}
}
```

Définition d’un variable de type classe/tableau

- Les classes sont de type comme les autres.
- La seul différence entre un type primitif et une classe/tableau est que un type primitif est une simple boite.
- Les tableaux sont des boites de boites.
- Les classes sont des définitions plus complexes de boites (la classe humain contient une boite pour l’age, pour le nom …).

Définition d’un variable de type classe/tableau

- Lors de la définition d’un tableau, on va donner combien de boite ce tableau peut accueillir.

- Un tableau se définit ainsi:
  - **<TypeSimple> tableau <NomTableau>[<Dimension>]**
  - Ex int tableau1[], ou int tableau2[][]

- L’initialisation de valeur a un tableau se fait ainsi:
  - int tableauEntier[] = {0,1,2,3,4,5,6,7,8,9};
  - int premiersNombres[][] = { {0,2,4,6,8},{1,3,5,7,9} };

Définition d’un variable de type classe/tableau

- Si un tableau n’est pas contruit via une initalisation, il est nécéssaire de préciser sa taille.
- Cette dernière se fait via l’opérateur **new:**
  - int [] a; a=new int [**4]; // 4 cases entières sont réservé.**

Définition d’un variable de type classe/tableau

- Les opérations sur les tableaux sont les suivantes:
  - **Soit un tableau X : int [] X=new int [5].**
- Le tableau X a 5 cases noté de 0 à 4.
- Il est possible de connaitre la taille d’un tableau X.length
- Il est possible d’écrire une valeur dans un tableau : X[2]=3;
- Il est possible de lire une valeur dans un tableau int a=X[1];

Définition d’une variable de type classe/tableau

- Une classe est un assemblage de « variable » appelé membres.
- Une structure est un assemblage de variables qui peuvent avoir différents types. Contrairement aux tableaux qui vous obligent à utiliser le même type dans tout le tableau, vous pouvez créer une structure comportant des variables de types long, char, int et double à la fois.
- Une définition de classe se fait en général dans un fichier portant le nom de la classe (la classe Toto seras écrite dans un fichier Toto.java)

Définition d’une variable de type classe/tableau

- Elle se définit par :

public class *Nom de la classe*

{

public « *type de variable* » « *nom de variable* »*;*

}

Définition d’une variable de type classe/tableau

- L’operateur new est aussi utilisé pour les classe.

```
Public class Humain
        { …
        ….
        }
Class Test
{
Public static void main(String [] args)
{
        // définition d’un humain
        Humain x;
        // initialisation d’un humain
        x=new Humain ()
}
}
```

Définition d'une variable de type classe/tableau

- Et on accède aux membres d'une classe via la notation « pointé »

```
Public class Humain
        {
        public int age;
        public String nom;
        }

Public Class Test
{
        Public static void main(String [] args)
{
        // définition d'un humain
        Humain x;
        // initialisation d'un humain
        x=new Humain ();
        x.age=3;
}
}
```

Définition d’une variable de type classe/tableau

- En pratique les tableaux et les classes sont initialisé via l’appel a new.
- Si l’initialisation est oublié, on dit que la valeur est « null » et sont emploie provoque des erreurs (essayer).
- « null » est une valeur particulière sans type signifiant « pas initialisé ».

Les constantes

- Les constantes sont des variables:
  - Appartenant a une définition de classe (utilisation de l'attribut static)
  - Qui ne sont pas modifiables (utilisation de l'attribut final)

```
Class Test
{
      public static final int MACONSTANTE=3;
}
```

Les variables

Saisie, affichage, affectation, conversion de type.

- Contenu:
  - Voir comment saisir des variables
  - Voir comment afficher les variables
  - Regarder la conversion de type
- Objectif:
  - Arriver a lire une ligne taper sur le clavier
  - Convertir une ligne représentant un entier … en entier

Les flux

- Un programme Java est un programme normal avec une gestion de flux :
  - Standard : flux d’entrée (le clavier), flux de sortie et flux d’erreur
  - Spécifique : flux réseau, flux fichier …

- Pour lire, nous utilisons le flux d’entrée qui est une variable de la classe System nommée in.

Lire un flux

- Nous allons crée un objet Scanner qui permet de lire des variables
- Nous l'initialisons avec le flux d'entrée
- Puis nous demandons le premier entier en utilisant la méthode nextInt();

```
Scanner sc = new Scanner(System.in);
System.out.println("Veuillez saisir un mot :");
String str = sc.nextLine();
System.out.println("Vous avez saisi : " + str);
```

Tester le flux

- Il est possible de tester si ce qui est tapez au clavier est bien un entier avec de demander la conversion en entier

```
Scanner sc = new Scanner(System.in);
If (sc.hasInt())
int i = sc.nextInt();
```

Sécuriser la lecture

- Enfin il est possible que la lecture soit problématique.
- Dans ce cas, nous gérons cela avec des « exceptions » qui sont la pour faire des traitements « exceptionnel » en cas de problèmes

```
Try{
Scanner sc = new Scanner(System.in);
If (sc.hasInt())
int i = sc.nextInt();
} catch (RuntimeException e)
{
}
```

Construction de base du langage

Les opérateurs

- Voir les différents opérateurs pour les variables.
- (Re) Voir les opérateurs sur les tableaux
- Voir les opérations sur les objets.

Les différents opérateurs

- Les opérateurs en Java ont une signification différentes en fonction des types de variables sur lequels ils portents.

- Nous avons :
  - Un opérateur de casting permettant de « transformer » un type en un autre
  - L’addition qui portent sur les entiers/floattant et chaine de caractères
  - La soustraction, la multiplication et la division qui portent sur les entiers/floattant
  - L’assignation qui est le fait de mettre une valeur dans une variable
  - Les opérateurs « booléen ».

Opérateur de casting

- Le casting est lorque l'on assigner un valeur d'un type à un autre.
- Le casting peut être implicite (le type va du plus petit ou plus grand):
  - byte -> short -> char -> int -> long -> float -> double
- Ou doit être expliciter avec l'opérateur de casting ()
  - double -> float -> long -> int -> char -> short -> byte

```
byte b=12;
int i=b;
byte b2=(byte) i;
```

Opérateur booléen

- Les opérateurs booléens sont des opérateurs dont le résultat est « vrai » ou « faux ».
- Nous trouvons des opérateurs prennant des entiers en paramètres:<, >, <=, >=, == (égalité) ou != différent.
- Il y a aussi des opérateurs prennant en paramètres des valeur booléenes: && (le ET), || (le ou) et le ! (non).
- Enfin il existe un opérateur d’égalité pour les objets, l’opérateur « equals »

Jouons un peu

- 2<5 est une expression correcte renvoyant vrai
- 3<1 est une expression correcte renvoyant faux.
- 2<5 && 3<1 est une expression correcte prennant « vrai » et « faux » pour les combiner avec un ET. Vrai et faux …. C'est faux.
  - En fait seul vrai et vrai donne un résultat vrai
  - Seul faux ou faux donne un résultat faux.
- ! (2<5 && 3<1) est une expression prennant « vrai » et « faux » et calculant l'inverse (avec l'opérateur !)

Les opérateurs sur les tableaux

Il existe deux opérateurs sur les tableaux:

- L'opérateur length qui renvoie la taille d'un tableau
- L'opérateur [] qui renvoie un élément d'un tableau

```
public class OperateurTableau {

        public static void main(String [] args)
        {
                int [] tableau=new int[3];
                System.out.println(tableau.length);
                System.out.println(tableau[2]);
        }
}
```

Les opérateurs objects

- Il y a trois opérateurs pour les classes:
  - L’opérateur new() qui reserve de la mémoire pour l’objet
  - L’opérateur de casting
  - L’opérateur instanceof
  - L’opérateur hashCode qui renvoie un identifiant pour un objet
  - L’opérateur de casting qui passe d’un objet a un autre
  - L’opérateur equals qui renvoie vrai si deux objet sont identiques
  - L’opérateur toString() qui renvoie une chaine de caractère pour un objet

Opérateur new

- L'opérateur new permet de réserver la mémoire d'un objet
- L'opérateur new s'écrie avec le nom de la classe de l'objet suivie (pour l'instant) de ()
- S'utilise ainsi : Point p=new Point();

```
public class Point {


        int x;
        int y;
        public Point()
        {

        }
}
```

L'opérateur de casting

- Les types structurés tel que les classes sont des Objects (une forme de super type des type structuré)

- On peut transfomer un type en object et vice versa
- Le casting peux échouer car les types ne sont pas castable;
- Il est utile parfois de gérer des « Objects » pour des questions de généricité

```
Object o=(Object)p1;
Point p3=(Point)o;
```

Erreur de casting

```
class Humain
{
String nom;
Humain()
{

}
}
….
Point p1=new Point();
Object o=(Object)p1;
Humain h=(Humain)o;
….
….
java.lang.ClassCastException class Point cannot be cast
to class Humain
```

L’opérateur instanceof

- L’opérateur instanceof permet de savoir si un object est d’un certain type

```
Point p1=new Point();
Object o=(Object)p1;

System.out.println(o instanceof Point);
System.out.println(o instanceof Humain);

true
false
```

Operateur hashCode

- L’opérateur hashCode renvoie un identifiant unique pour un objet
- Eclipse est capable de générer cet opérateur
- L’opérateur s’utilise ainsi : Point p=….; p.hashCode();

```
public int hashCode() {
            final int prime = 31;
            int result = 1;
            result = prime * result + x;
            result = prime * result + y;
            return result;
      }
```

Opérateur hashCode

- P1==p2 renvoie faux (il s'agit de deux objet « différent »)
- P1.hashCode()==p2.hashCode() renvoie vraie car les valeurs de x et y sont identiques.

```
public static void main(String [] args)
{
Point p1=new Point(); p1.x=2; p1.y=2;
Point p2=new Point(); p2.x=2; p2.y=2;
System.out.println(p2==p1);
System.out.println(p2.hashCode()==p1.hashCode());


}
```

Opérateur equals

- L’égalité == ne marche pas pour les objets.
- L’opérateur d’égalité est l’opérateur equals pour les objets.
- Cet opérateur est très bien géré par Eclipse

```
public boolean equals(Object obj) {
			if (this == obj)
				return true;
			if (obj == null)
				return false;
			if (! this != obj.getClass())
				return false;
			Point other = (Point) obj;
			if (x != other.x)
				return false;
			if (y != other.y)
				return false;
			return true;
		}
Point p2, p1;
System.out.println(p2.equals(p1));
```

Opérateur toString

- L’opérateur toString() permet d’afficher joliment un objet.

```
public String toString() {
        return "Point [x=" + x + ", y=" + y + "]";
        }
```

Les sélections alternatives

- Contenu:
  - Selection simple If Then
  - Selection avec cas par défaut: If Then Else
  - Sélection multiple: switch (condition) case (…)
- Objectif:
  - Connaitre et posséder les If Then Else

Les sélections alternatives

- Pour l’instant nous n’avons fait que des programmes executant les instructions les unes après les autres SANS condition.

- Il est possible de mettre en place des conditions afin de faire des traitements différencié. Par exemple un logiciel permettant d’avoir de l’argent dans un distributeur. **Si** le solde de la personne est positif **alors** la machine distribue de l’argent.

Structure de IfThen

- La construction précédente est traduite en Java ainsi:
- If (*Condition booléene*) instruction
- Cela doit être lue en si la condition booléene est vrai alors on execute l'instruction.

Exemple de code

```
int i = 10;

if (i < 0)
System.out.println("le nombre est négatif");
else
System.out.println("le nombre est positif");
```

Structure de IfThenElse

- Il est aussi possible de faire une structure Si condition Alors instruction Sinon Instruction.
- La structure s'écrit en Java :
  - If (*condition booléene*) *instruction1* else *instruction2.*
  - A comparer avec If (*condition booléene*) *instruction1*

Exemple de code

```
int i = 0;
if (i < 0)
{
System.out.println("Ce nombre est négatif !");
}
else
{
if(i == 0)
System.out.println("Ce nombre est nul !");

else
System.out.println("Ce nombre est positif !");

}
```

Selection multiple

- Il est possible d'enchainer les If Then Else ainsi:
  - If (condition 1) { }; if (condition 2) { } ….
- Cet enchainement peut s'écrire pour plus de lisiblité
  - Switch (condition)

    Case X // cas 1

    Case Y // cas 2

Exemple de code

```
int note = 10; //On imagine que la note maximale est 20

switch (note)
{
case 0:
System.out.println("Ouch !");
break;
case 10:
System.out.println("Vous avez juste la moyenne.");
break;
case 20:
System.out.println("Parfait !");
break;
default:
System.out.println("Il faut davantage travailler.");
}
```

Equivalence

- Un If Then Else est un If Then avec la clause Else vide
- Ou autrement dit, un If Then Else peut s'écrire avec 2 If Then
- Les structure switch sont des If Then enchainé

C'est ce qu'on appel le « sucre » syntaxique!

Les structures de contrôle

Les boucles itératives

- Contenu:
  - Boucle for
  - Boule while et do
- Objectif:
  - Connaitre et posséder les boucle while

Boule while

- C’est la boucle tant que qui doit ce lire
  - Tant que la condition est vrai alors faire un bloc d’instruction

```
while (/* Condition */)
{
//Instructions à répéter
}
```

Exemple de code

```
Int i=0;
While (i<10)
{
System.out.println(i);
i=i+1;
}
```

Boucle faire tant que

- C’est la boucle « sœur » de la boucle while:

```
do{
//Instructions
}while(a < b);
```

- Contrairement à la boucle while, la boucle do s’execute au moins une fois.

Exemple de code

```
Int i=0;
Do
{
System.out.println(i);
i=i+1;
} while (i<10);
```

Boucle for

- La boucle for est à la fois la plus « simple » et peu devenir la plus complexe.

- Elle se compose de trois partie:
  - La première est la déclaration/initialisation des variables
  - La deuxième est la condition d'arret de la boucle
  - La troisième est l'action a effectuer a chaque tour de boucle.

La boucle for en version simple

```
for(int i = 1; i <= 10; i++)
{
System.out.println("Voici la ligne "+i);
}
for(int i = 10; i >= 0; i--)
System.out.println("Il reste "+i+" ligne(s) à écrire");
```

La boucle for en version complexe

```
// ce type de boucle est à proscrire
for(int i = 0, j = 2; (i < 10 && j < 6); i++, j+=2){
System.out.println("i = " + i + ", j = " + j);
}
```

For Each

- Java possède une boucle for each qui est un équivalent de la boucle for.
- La syntaxe est for (<type> nom: tableau ou iterable)

```
Int tab=new int[10];
For (int i=0;i<tab.length;i++) System.out.println(tab[i]);
// est équivalent à
For (int valeur:tab) System.out.println(valeur).
```

Imbrication des instructions.

- Contenus
  - Voir la notion d’instruction multiple
  - Comprendre la notion de porté de variable
- Objectifs:
  - Voir la notion de porté

Imbrication des instructions

- Un bloc d’instruction commence par le symbole { et se termine par le symbole }.
- Toutes les instructions de contrôle portent soit sur une instruction soit sur un bloc de variable.
- Par exemple:
  - If (condition) instruction // une seule instruction
  - If (condition) **{ instruction1; instruction2; …. Instruction n; }**

Porté des variables

- Les variables déclarés entre deux accolades ne sont connues qu'entre ces deux accolade.

```
{
int a=0;
if (a<10){System.out.println(a);
}
}
a = 1; // le compilateur refuse. La variable a n'est plus déclaré ici
```

Bonne pratique

- Il est licite de déclarer des blocs d’instructions en dehors de structure de contrôle … mais cela n’améliore pas la lisibilité
- Il est possible de déclarer une instruction dans une structure de contrôle mais cela n’est pas très lisible. On préféreras toujours définir une bloc d’instruction même pour une instruction
  - **int a = 1;{a=a+2;{a=a+3;}}if (a==2) if (a==4) a=a+1;else a++; illisible**
  - **int a = 1;a=a+2;a=a+3;if (a==2) {if (a==4) {a=a+1;}else {a++;}} Ok**

Les structures de contrôle

Les commentaires.

- Contenu:
  - Voir les différents types de commentaires
  - Générer une javadoc
- Objectif:
  - Connaitre l’utilisation des commentaires
  - Connaitre quelques tag javadoc

Commentaires

- Il y a deux syntaxes pour l’utilisation des commentaires
  - // qui permet de faire des commentaires sur une lignes
  - /*

  qui permet de faire des commentaires sur plusieurs lignes

  */

- La syntaxe « classique » en Java consiste a utiliser:
  - La notation // pour des annotations
  - La notation /**

  *pour les autres commentaires

  */

Quoi commenter?

- En pratique on commente les éléments précédé par le mot clef « public »
- La raison est que ce sont les « points » d’entrée des programmes.
- De ce fait la façon de faire de la documentation est « standardisé » en Javavia l’outil Javadoc et les tag Javadoc

Tag JavaDoc

```
/**
 * Renvoie l'addition de deux entiers
 * @param a le premier entier
 * @param b le deuxième
 * @return la somme des deux
 */
protected static int addition(int a,int b)
{
      return a+b;
}
```

Introduction à la programmation avec Java

- Les fondements de la programmation
- Les constructions de base d'un programmes
- Tests Logiciels
- Pourquoi faire des tests ?
  - Présentation des différents types de tests : tests unitaires, fonctionnels, de robustesse et de performance.
  - Quels tests lancer et quand ?
  - Utilité des objets "Mock" et "Fake" durant les tests unitaires. Couverture des tests unitaires.
- Bonne Pratiques de conception d'une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d'un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Pourquoi faire des tests

- L’idée est de montrer ce qu’est un bug, son implication dans le développement logiciel

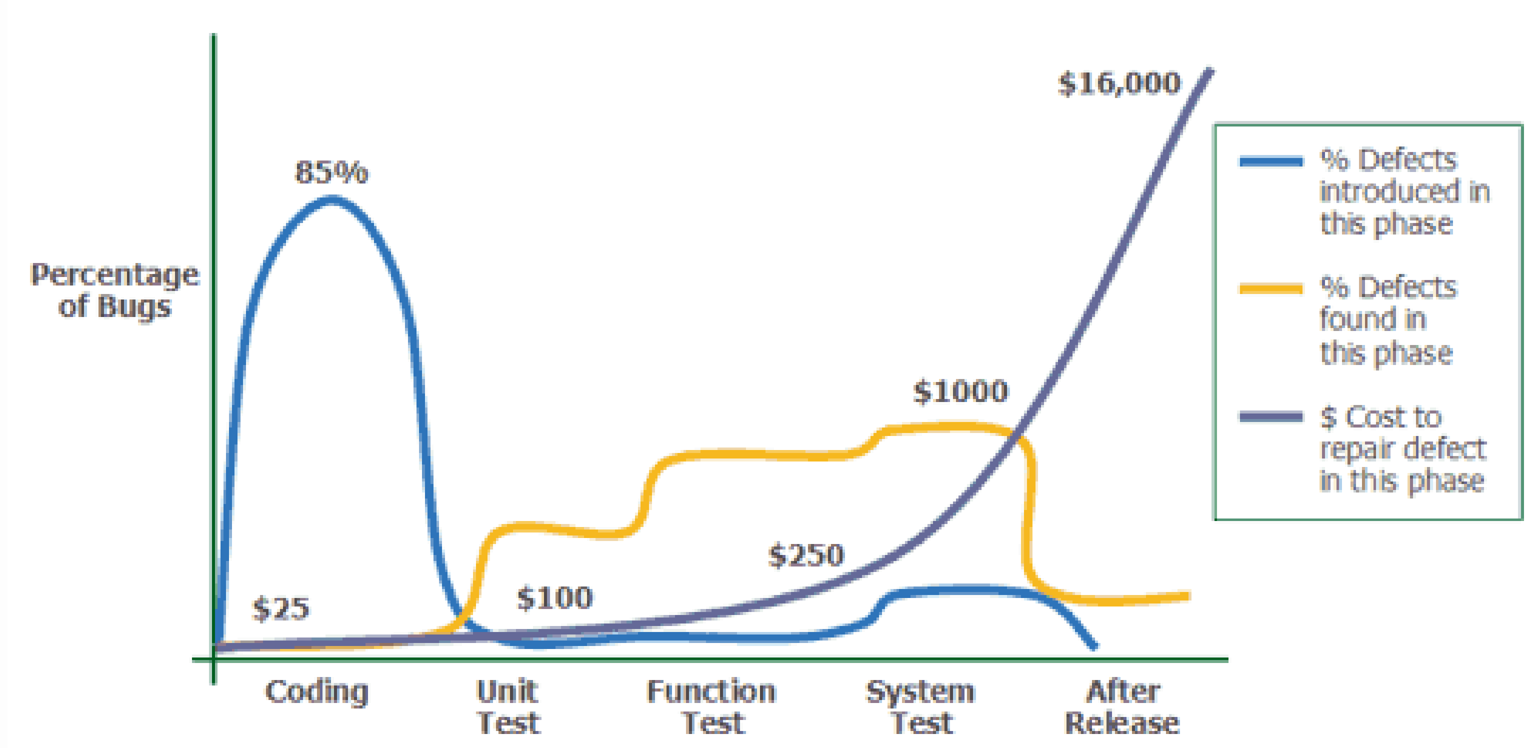
Pourquoi faire des tests?

Quelques banalités:

- Tester c’est douter, c’est fait pour les mauvais.
- On fait tester par le client, cela n’est pas mon problème.
- On code le projet, et on teste ensuite.

En réalité, plus un bug est vue tard plus, il est cher à corriger (hors mise de retard sur un marché, et hors perte due au bug)

Implications sur la qualité logiciel



source: Applied Software Measurement, Capers Jones

Ce schéma ne prend pas le coût du retard de mise sur le marché

Pourquoi faire des tests?

Mais la question est qu’est ce qu’un bug?

Un bug est une différence entre un résultat attendue et un résultat obtenue.

- Le bug « simple » est un bug de code (le logiciel ne fonctionne pas)
- Un bug fonctionnel, le logiciel fonctionne pas par sur tel ou tel fonctionnalité
- Un bug de receuil de besoin, le logiciel ne répond pas à mon besoin

Qu’est-ce qu’un projet?

Si un logiciel est une réponse technique à un besoin.

Alors un logiciel implique plusieurs projets:

- Le chantier de réalisation ou « release » / « build »
- Le chantier de maintenance ou « run », avec des sous éléments

- Le cout d’un bug (et l’absence de tests) est plus important dans le cadre de chantier de maintenance car l’application est déjà en production. (on est déjà sur la courbe la plus haute)
- De plus un bug doit se comprendre dans le cadre de la gestion d’un projet

Qu'est-ce qu'un projet?

Les projets se découpe en

- Avant-projet
  - Etude d'opportunité: Est-ce que le projet est opportun?
  - Etude d'impact: Est-ce que ce dernier engendre un cout?
  - Choix de solutions: Est-ce que le projet est faisable et si oui comment?
- Création
  - Conception : Comment le projet doit être fait?
  - Réalisation : Instanciation du projet
- Recette
  - Technique: Est-ce que techniquement le projet fonctionne?
  - Fonctionnel: Est-ce que fonctionnellement le projet fonctionne?
  - Non fonctionnel: Est-ce que ce dernier est utilisable?
- Après projet
  - Bilan: Est-ce que le projet répond au besoin?
  - Clôture du projet : Fermeture du projet

Qu'est-ce qu'un projet?

Chaque phase d'un projet implique des tests qui sont de la responsabilité des équipes de DEV ou pas. Il y a par phase des tests associés.

Les tests sont fonctions de l'aspect du logiciel qui est testé.

Enfin les tests sont fait avec le code du logiciel, le logiciel (sans le code) et sans ni le code ni le logiciel.

Qu'est-ce qu'un test?

Pour le développement

- Les développeurs ont a minima à leur charge:
- Les test unitaires (est ce que les fonctions marchent)
- Les tests d’intégrations (est ce que l’assemblage des fonctions marchent)
- Une partie des tests fonctionnels (est ce qu’une fonctionnalité marche).
- Ces tests sont fait en fonction des entreprises soit en boite noir (sans le code) soit en boite blanche (avec le code).

Qu'est-ce qu'un test?

- La définition de la qualité logiciel peut être étendue:
  - De mener un projet de Agilité (vue comme une succession de projet)
  - De partager l'information entre les développeurs (technique) et les testeurs (plus proches du fonctionnel)
  - Représente *in fine* les spécifications réelles du logiciel.
  - De garantir les évolutions du logiciels en évitant les régressions.
- Une bonne définition d'un test est :
  - Un test est une exécution sur un jeu de donnée précis avec une méthodologie
  - Un test a un but qui permet de vérifier ou de valider une fonctionnalité
  - La qualité logicielle est le marqueur de la bonne/mauvaise avancé d'un projet.
- Enfin, la qualité logicielle est un arbitrage budgétaire impliquant soit un surcout (test inutile) soit un gain de cout (aide aux développeurs, « early bug avoidance », aide à la documentation, validation de l'usage).

Qu'est-ce qu'un bug/anomalie?

- Une anomalie (résultat d'un bug) est caractérisée par plusieurs couts:
  - Cout pour détecter l'anomalie
  - Cout pour résoudre l'anomalie
  - Valeur négative de la valeur d'usage du logiciel
  - Cout d'image pour la société
- Et permet d'avoir de la valeur
  - Justification d'un chantier de maintenance
  - Anomalie en moins (syndrome de l'arbre « bug » qui cache la forêt de « bug »).
  - Non-respect d'une norme contraignante
  - Approche nouvelle/fonctionnalité étendue d'un logiciel

Cout d'une anomalie (étude NIST)

Cout de réparation		Temps de détection				
		Besoin	Architecture	Construction	PreProduction	Mise sur le marché
Moment de création du bug	Besoin	X 1	X 3	X 5-10	X 10	X10 – X 100
	Architecture		X 1	X 10	X 15	X 25 – X 100
	Construction			X 1	X 10	X10 – X25

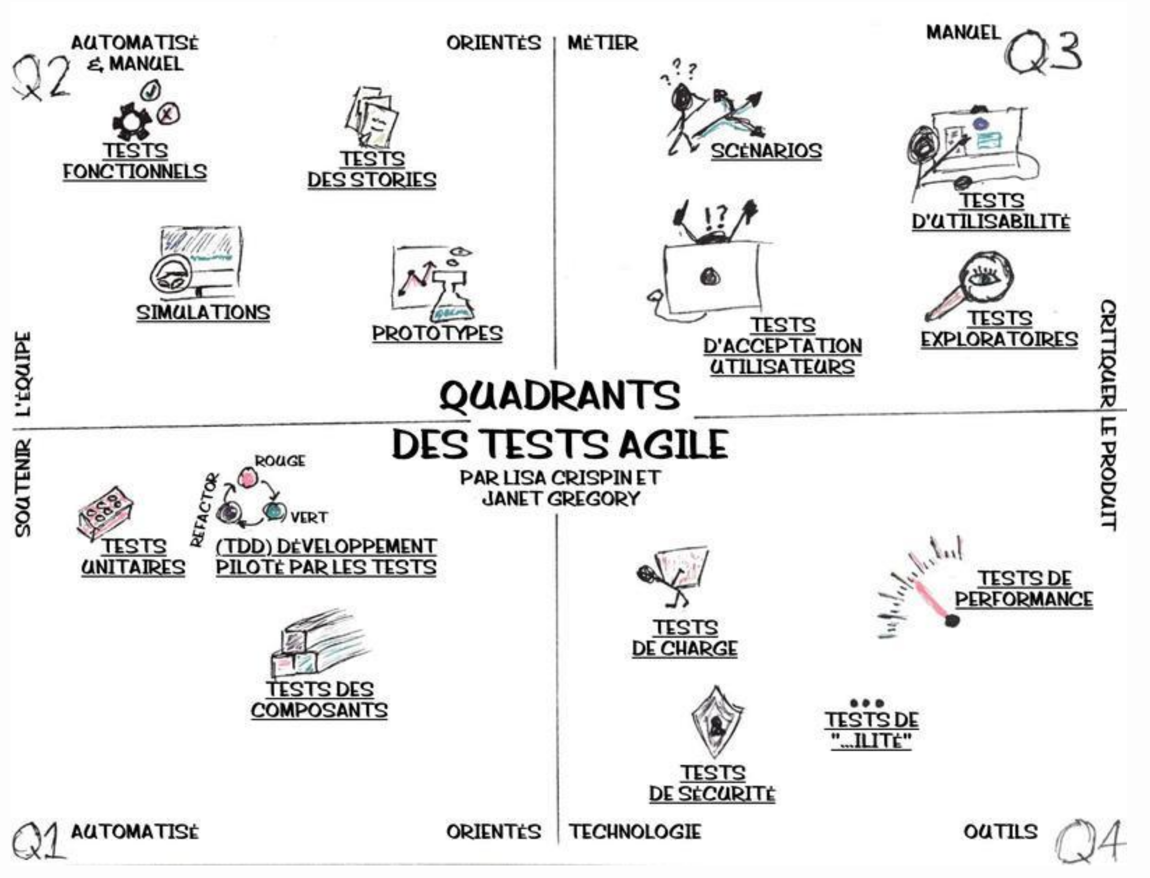
Le Magic Quadrant

- Le Magic Quadrant est le modèle proposé par Lisa Crispin pour mettre en avant la qualité logiciel.

Le Magic Quadrant

- Il s’agit du modèle pratique proposé par Lisa Crispin et Janet Grégory pour mettre la qualité logiciel au centre du développement Agile.
- Les grandes valeurs du modèle sont que les tests vont aider les co-équipiers et étant le garant d’une bonne gestion de projet.
- Le produit seras testé au regard des métiers et des caractéristiques non fonctionnelles (performance)

Le quadrant Agile



Quadrant 1 et 2

Quadrant Agile I – Dans ce quadrant, on se concentre sur la qualité interne du code, avec des cas de tests pilotés par la technologie et mis en oeuvre pour aider l'équipe. On y trouve :

- Tests unitaires
- Tests d'intégration ou de composants

Quadrant Agile II – Dans ce quadrant, les cas de tests sont pilotés par le métier et sont mis en oeuvre pour aider l'équipe. Ce quadrant se concentre sur les besoins.

Les types de tests menés dans cette étape sont :

- Tests avec des exemples de scénarios et workflows possibles
- Tests d'Expérience Utilisateur, par exemple avec des prototypes
- Tests binômés métiers/développeurs

Quadrant 3 et 4

Quadrant Agile III – Ce quadrant donne du feedback aux quadrants I et II. Les cas de tests peuvent être utilisés pour établir des tests automatisés. Dans ce quadrant, plusieurs revues d'itération sont menées, ce qui construit le confiance dans le produit. Les types de tests réalisés dans ce quadrant sont :

- Tests d'utilisabilité
- Tests exploratoires
- Tests binômés avec les clients
- Tests collaboratifs
- Tests d'acceptation utilisateurs

Quadrant Agile IV – Ce quadrant se concentre sur les exigences non-fonctionnlles telles que la performance, la sécurité, la stabilité, ... Dans ce quadrant, l'application est fabriquée pour offrir des qualités non fonctionnelles et la valeur attendue.

- Tests non fonctionnels tels que les tests de performance et de robustesse
- Tests de sécurité en rapport avec l'authentification et le piratage
- Tests d'infrastructure
- Tests de migration de données
- Tests de scalabilité
- Tests de charge

Quelques guides

La plupart des projets commenceraient par des tests Q2, parce que c'est là que l'on trouve des exemples qui se transforment en spécifications et en tests qui pilotent le codage, ainsi que des prototypes et autres.

Les tests Q3 et Q4 exigent qu'une partie du code soit écrite et déployable, mais la plupart des équipes itèrent rapidement dans les quadrants, travaillant par petits incréments.

Ecrire un test pour un petit morceau d'une fonctionnalité, écrire le code, une fois le test réussi, peut-être automatiser d'autres tests, mener des tests exploratoires, mener des tests de sécurité ou de charge, peu importe, puis ajouter le petit morceau suivant de fonctionnalité et recommencer l'ensemble du processus

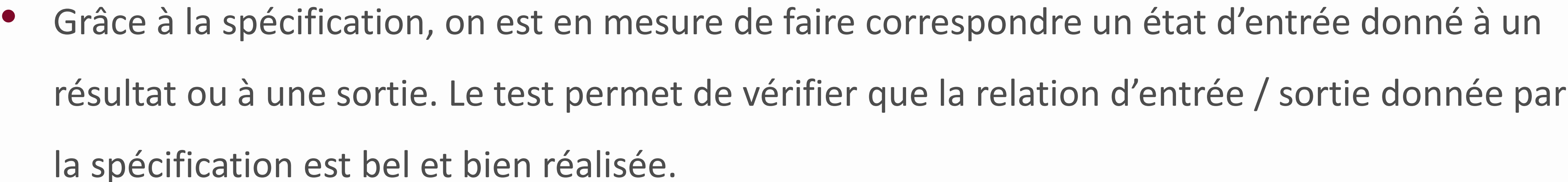
Tests automatisés avec le framework JUnit

Le besoin d'un framework de test. JUnit.

- Qu'est ce qu'un framework de test?
- Qu'est ce que JUnit

Besoin d'un framework de test

- On écrit un test pour confronter une réalisation à sa spécification. Le test définit un critère d’arrêt (état ou sorties à l’issue de l’exécution) et permet de statuer sur le succès ou sur l’échec d’une vérification.



- Grâce à la spécification, on est en mesure de faire correspondre un état d’entrée donné à un résultat ou à une sortie. Le test permet de vérifier que la relation d’entrée / sortie donnée par la spécification est bel et bien réalisée.

Besoin d'un framework de test

- Trouver les erreurs rapidement:
  - La méthode XP préconise d'écrire les tests en même temps, ou même avant la fonction à tester (Test Driven Development). Ceci permet de définir précisément l'interface du module à développer. Les tests sont exécutés durant tout le développement, permettant de visualiser si le code fraîchement écrit correspond au besoin.
- Sécurise la maintenance:
  - Lors d'une modification d'un programme, les tests unitaires signalent les éventuelles régressions
- Documentation du code:
  - Il est très utile de lire les tests pour comprendre comment s'utilise une méthode. De plus, il est possible que la documentation ne soit plus à jour, mais les tests eux correspondent à la réalité de l'application.

Besoin d'un framework de test

On définit généralement 4 phases dans l'exécution d'un test unitaire :

- **Initialisation** (setUp) : définition d'un environnement de test complètement reproductible
- **Exercice** : le module à tester est exécuté
- **Vérification** (Assert) : comparaison des résultats obtenus avec un vecteur de résultat défini. Ces tests définissent le résultat du test : SUCCÈS (SUCCESS)ou ÉCHEC (FAILURE). On peut également définir d'autres résultats comme EVITE (SKIPPED).
- **Désactivation** (fonction tearDown) : désinstallation pour retrouver l'état initial du système, dans le but de ne pas polluer les tests suivants. Tous les tests doivent être indépendants et reproductibles unitairement (quand exécuté seul).

Besoin d'un framework de test

- La famille des XUnit offrent des outils de création de test pour un langage particulier :
  - JSUnit, QUnit et Unit.js pour JavaScript ;
  - JUnit et TestNG pour Java ;
  - cppunit pour C++ ;
  - CUnit pour C ;
  - OCunit pour Objective C ;
  - NUnit pour .NET ;

Besoin d'un framework de test

- … ou un domaine d'application particulier
  - DBUnit pour les applications bases de données
  - JUnit pour Android
  - JUnit avec selenium pour les applications Web.

Junit

- Framework de test unitaire en Java écrit par Kent Beck et Erich Gamma(97).
- Open Source, il est issue la série de xUnit (Ici le x est le langage Java).
- Intégrée à BlueJ, Eclipse ou Netbeans.
- Existe en trois versions JUnit 3 (sans annotations, et quasi inutilisé), JUnit 4 (avec annotations) et maintenant JUnit 5
- Bibliothèque de base pour la création de test

JUnit

- Une série de test JUnit est écrite dans un fichier classe contenant une liste de test. Le paradigme est que la classe est une TestSuite et le test, un test case.
- Les classes de tests sont exécuté par une classe Runner dont le rôle est d'enchainer l'execution des tests et en donner le résultat.

JUnit5

- JUnit 5 est une réécriture complète de l'API contenue dans des packages différents de ceux de JUnit 4.avec un lot de nouvelles fonctionnalités :
  - les tests imbriqués
  - les tests dynamiques
  - les tests paramétrés qui offrent différentes sources de données
  - un nouveau modèle d'extension
  - l'injection d'instances en paramètres des méthodes de tests

- La version 5 de JUnit est composée de trois sous-projets :
  - JUnit Platform : propose une API permettant aux outils de découvrir et exécuter des tests. Il définit une interface entre JUnit et les clients qui souhaitent exécuter les tests (IDE ou outils de build par exemple)
  - JUnit Jupiter : propose une API reposant sur des annotations pour écrire des tests unitaires JUnit 5 et un TestEngine pour les exécuter
  - JUnit Vintage : propose un TestEngine pour exécuter des tests JUnit 3 et 4 et ainsi assurer une compatibilité ascendante

JUnit

- Création d'une classe contenant un test utilise l’annotation @Test

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class MonTest {

  @Test
  public void simpleTest() {
    System.out.println("simpleTest");
    Assertions.assertTrue(true);
  }
}
```

JUnit

- La fonction assertTrue permet dire que le test est reussie si le résultat est « vraie »

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class MonTest {

  @Test
  public void simpleTest() {
    System.out.println("simpleTest");
    Assertions.assertTrue(true);
  }
}
```

Display Name

- Les classes et les méthodes de tests peuvent avoir un libellé qui sera affiché par les tests runners ou dans le rapport d'exécution des tests. Ce libellé est défini en utilisant l'annotation @org.junit.jupiter.api.DisplayName. Elle n'attend qu'un seul attribut obligatoire qui précise le libellé.

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

@DisplayName("Ma classe de test JUnit5")
public class MonTest {

  @Test
  @DisplayName("Mon cas de test")
  void premierTest() {
    // ...
  }
}
```

Annotation JUnit

Une classe de test JUnit peut avoir des méthodes annotées pour définir des actions exécutées durant le cycle de vie des tests. Le cycle de vie d'un test peut être enrichi grâce à quatre annotations utilisées sur des méthodes pour réaliser des initialisations ou du ménage :

- @BeforeAll : exécutée une seule fois avant l'exécution du premier test de la classe
- @BeforeEach : exécutée avant chaque méthode de tests
- @AfterEach : exécutée après chaque méthode de tests
- @AfterAll : exécutée une seule fois après l'exécution de tous les tests de la classe

Les annotations

```
public class MonTest {

  @BeforeAll
  static void initAll() {
    System.out.println("beforeAll");
  }

  @BeforeEach
  void init() {
    System.out.println("beforeEach");
  }

  @AfterEach
  void tearDown() {
    System.out.println("afterEach");
  }

  @AfterAll
  static void tearDownAll() {
    System.out.println("afterAll");
  }

  @Test
  void simpleTest() {
    System.out.println("simpleTest");
    Assertions.assertTrue(true);
  }

  @Test
  void secondTest() {
    System.out.println("secondTest");
    Assertions.assertTrue(true);
  }
}
```

Cycle de vie d'un test

- Une méthode annotée avec @BeforeAll sera exécutée avant l'exécution de la première méthode de tests. Avec le cycle de vie par défaut des instances de tests, il est obligatoire qu'une méthode annotée avec @BeforeAll soit statique.
- Une méthode annotée avec @AfterAll est exécutée après l'exécution de toutes les méthodes de tests de la classe.
- Une méthode annotée avec @BeforeEach sera exécutée avant chaque exécution d'une méthode de tests. Elle ne peut pas être statique sinon une exception de type JUnitException est levée à l'exécution.
- Une méthode annotée avec @AfterEach sera exécutée après chaque exécution d'une méthode de tests. Elle ne peut pas être statique sinon une exception de type JUnitException est levée à l'exécution.

Les assertions

- Les assertions ont pour rôle de faire des vérifications pour le test en cours. Si ces vérifications échouent, alors l'assertion lève une exception qui fait échouer le test.
- Les assertions classiques permettent de faire des vérifications sur une instance ou une valeur ou effectuer des comparaisons. La classe org.junit.jupiter.Assertions contient de nombreuses méthodes statiques qui permettent d'effectuer différentes vérifications de données. Ces assertions permettent de comparer les données obtenues avec celles attendues dans un cas de test.

Les assertions

Egalité	Nullité	Exceptions
assertEquals()	assertNull()	assertThrows() pour vérifier la lever d'une exception durant le test
assertNotEquals()	assertNotNull()	
assertTrue()		
assertFalse()		
assertSame()		
assertNotSame()		

assertEquals

- L'assertion assertEquals permet de vérifier que la valeur actuelle et la valeur attendue soient égales.
- La méthode assertEquals() de la classe Assertions possède de nombreuses surcharges pour différents types de données et éventuellement un message sous la forme d'une chaîne de caractères.
  - public static void assertEquals(xxx expected, xxx actual)
  - public static void assertEquals(xxx expected, xxx actual, String message)

```
@Test
 void verifierEgalite() {
   Dimension sut = new Dimension(801, 601);
   Assertions.assertEquals(new Dimension(800, 600), sut,
"Dimensions non egales");
 }
```

assertTrue

- L'assertion assertTrue permet de vérifier que la condition fournie est vraie.
- La méthode assertTrue() de la classe Assertions possède plusieurs surcharges pour fournir la condition sous la forme d'un booléen et éventuellement un message :
  - public static void assertTrue(boolean condition)
  - public static void assertTrue(boolean condition, String message)

```
@Test
 void verifierTrue() {
   boolean bool = true;
   Assertions.assertTrue(bool);
   Assertions.assertTrue(MonTest.getBooleen(), "Booleen different de true");
 }

 static boolean getBooleen() {
   return false;
 }
```

Desactivation des tests

- L'annotation @org.junit.jupiter.api.Disabled permet de désactiver un test.
- Il est possible de fournir une description optionnelle de la raison de la désactivation
- L'annotation @Disabled peut être utilisée sur une méthode ou sur une classe. L'utilisation sur une méthode désactive uniquement le test concerné.

```
@Test
  @Disabled("A écrire plus tard")
  void monTest() {
    fail("Non implémenté"); /* fait echouer le test si Disabled
n’est pas présent */
  }
```

Runner

- Le Runner est le moteur d'execution des tests
- Il est présent par défaut, mais peut être spécifié.
- Le moteur par défaut est JUnitPlatform

```
JUnit5
  .settings
  src
    main
    test
      java
        com
          howtodoinjava
            junit5
              examples
                packageA
                  ClassATest.java
                packageB
                  ClassBTest.java
                packageC
                  ClassCTest.java
                JUnit5TestSuiteExample.java
  target
  .classpath
  .project
  pom.xml
```

```
@RunWith(JUnitPlatform.class)
@SelectPackages("com.howtodoinjava.junit5.examples.packageA")
public class JUnit5TestSuiteExample
{
}
```

Runner

- On peut demander a exécuter plusieurs série de tests dans des packages
- Ou plusieurs classes

```
JUnit5
  .settings
  src
    main
    test
      java
        com
          howtodoinjava
            junit5
              examples
                packageA
                  ClassATest.java
                packageB
                  ClassBTest.java
                packageC
                  ClassCTest.java
                JUnit5TestSuiteExample.java
  target
  .classpath
  .project
  pom.xml
```

```
@RunWith(JUnitPlatform.class)
@SelectPackages({"com.howtodoinjava.junit5.examples.packageA","com.howtodoinjava.junit5.exampl
es.packageB"})
public class JUnit5TestSuiteExample
{
}

@RunWith(JUnitPlatform.class)
@SelectClasses( { ClassATest.class, ClassBTest.class, ClassCTest.class } )
public class JUnit5TestSuiteExample
{
}
```

Tag

- Il est possible de regrouper des tests sous formes de catégorie via des Tags. Ces Tags seront ensuite filtré via le Runner

```
@Test
@Tag("IntegrationTest")
public void testAddEmployeeUsingSimpelJdbcInsert() {
}

@Test
@Tag("UnitTest")
public void givenNumberOfEmployeeWhenCountEmployeeThenCountMatch() {
}
```

```
@RunWith(JUnitPlatform.class)
@SelectPackages(“be.etnic.myjob.test.dao")
@IncludeTags("UnitTest")
public class EmployeeDAOUnitTestSuite {
}
```

Le Mock

- Qu’est ce qu’un Mock
- Comment implémenter un Mock

Mockito

- Dans la création d’un test il est parfois complexe d’avoir des objets complets a tester:
  - Un objet qui revient d’une base de données. Comment le tester sans base de données.
  - L’authentification avec LDAP. Comment tester sans avoir a monter un LDAP.
- Le truc est de créer des objets représentant les éléments qui manque (un faux LDAP, une fausse base de données …)
- Les Mock peuvent s’appeler des bouchons ou encore des stubs.

Mock

- Un test unitaire doit tester la fonctionnalité de manière isolée. Les effets secondaires d'autres classes ou du système doivent être éliminés si possible pour un test unitaire.
- Cela peut être fait en utilisant des remplacements de test (Mock) pour les dépendances réelles. Les Mockpeuvent être classés comme suit:
  - Un objet factice est passé mais n'est jamais utilisé, c'est-à-dire que ses méthodes ne sont jamais appelées. Un tel objet peut par exemple être utilisé pour remplir la liste de paramètres d'une méthode.
  - Les faux objets ont des implémentations fonctionnelles, mais sont généralement simplifiés. Par exemple, ils utilisent une base de données en mémoire et non une vraie base de données.
  - Une classe de stub est une implémentation partielle d'une interface ou d'une classe dans le but d'utiliser une instance de cette classe de stub lors des tests. Les stubs ne répondent généralement à rien en dehors de ce qui est programmé pour le test. Les stubs peuvent également enregistrer des informations sur les appels.
  - Un objet fictif est une implémentation factice pour une interface ou une classe dans laquelle vous définissez la sortie de certains appels de méthodes. Les objets fantaisie sont configurés pour exécuter un certain comportement lors d'un test. Ils enregistrent généralement l'interaction avec le système et les tests peuvent le valider.

Mock

- Vous pouvez créer des objets fictifs manuellement (via du code) ou utiliser un framework fictif pour simuler ces classes. Les frameworks factices vous permettent de créer des objets fantaisie lors de l'exécution et de définir leur comportement.

- L'exemple classique d'un objet fictif est un fournisseur de données. En production, une implémentation permettant de se connecter à la source de données réelle est utilisée. Mais pour tester, un objet factice simule la source de données et garantit que les conditions de test sont toujours les mêmes.

- Ces objets fictifs peuvent être fournis à la classe qui est testée. Par conséquent, la classe à tester devrait éviter toute dépendance difficile aux données externes.

- Les structures mock ou mock permettent de tester l'interaction attendue avec l'objet mock. Vous pouvez, par exemple, valider que seules certaines méthodes ont été appelées sur l'objet fictif.

Mockito

- Mockito est un de ses framework qui permet de créer des Mock sur un objet:
- La méthode mock permet de créer un objet a partir d’une classe ou d’une interface.
- Il est aussi possible de définir le retour d’une fonction via la fonction When

```
        Money toTest;

        @BeforeEach
        public void init() {
                toTest=Mockito.mock(Money.class);


org.mockito.Mockito.when(toTest.getAmount()).thenReturn(10);
          }
```

Introduction à la programmation avec Java

- Introduction
- Les constructions de base d’un programmes
- Tests Logiciels
- Bonne Pratiques de conception d’une application
  - Découpage en couches (données, métier, présentation).
  - Présentation des enjeux d'un développement d'entreprise.
  - Introduction à l'écosystème JAVA (JEE, Spring, Hibernate, JSF ou Struts...).
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d’un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Bonne Pratiques de conception d’une application

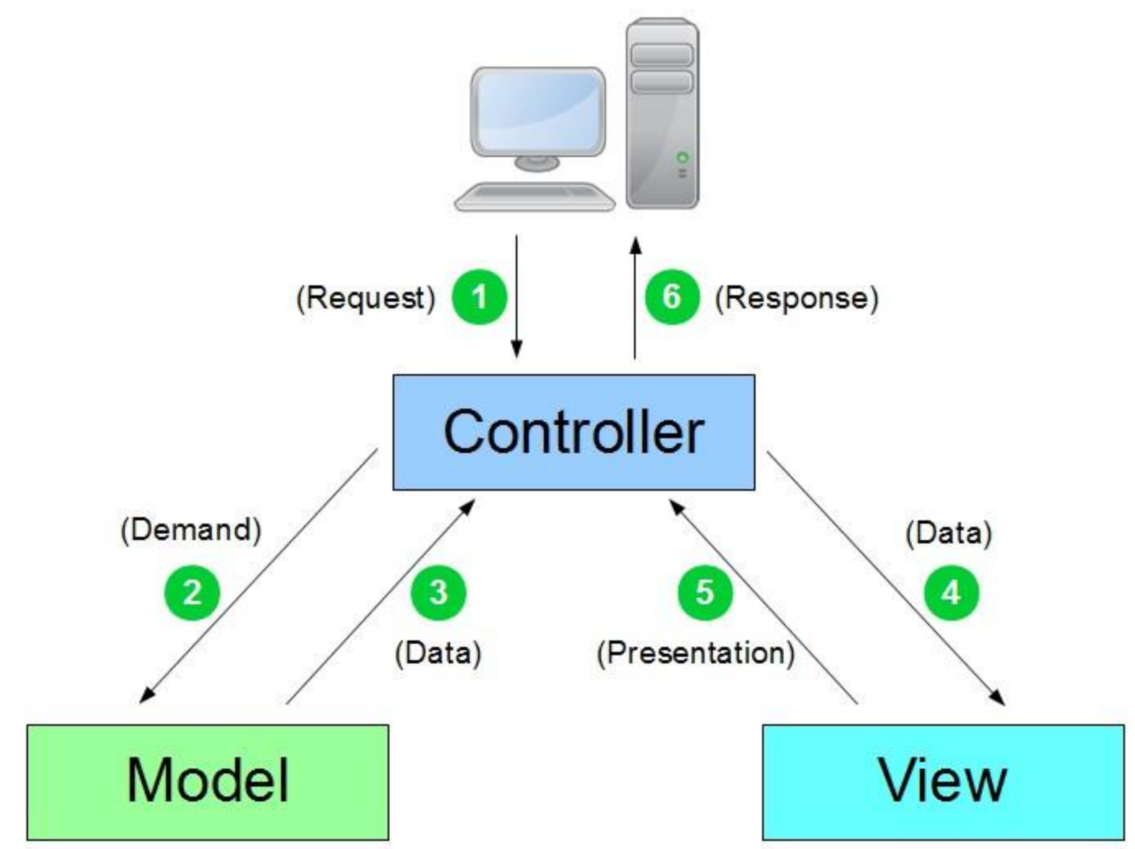
Découpage en couches (données, métier, présentation).

- Comprendre la conception MVC

sss

- Le standard Java EE n'imposent aucun model d'application, c'est-à-dire qu'on peut développer n'importe comment.
  - Le problème c'est que si on développe n'importe comment notre code va être mal organisé et il devient très vite difficile de retrouver un bout de code ou une fonction qu'on veut modifier
- La tendance est d'utiliser les bonnes pratiques de développement qu'on appelle les Patrons de conception ou Design Pattern
- Le modèle MVC découpe littéralement l'application en couches distinctes et de ce fait impacte très forcement l'organisation du code. MVC signifie Modèle – Vue – Contrôleur.

Le Modèle MVC



Modèle MVC

- (1)Le visiteur envoie sa requête http, la transmet au serveur d'application qui la transmet directement à la partie de notre code qu'on appelle le Contrôleur.
- (2) Le contrôleur lui il fait le routage de l'information en décidant qui va récupérer l'information et ensuite la traiter, il appelle en effet le Modèle qui contient les informations structurées et qui va effectuer des calculs ou traitement sur ces informations et les renvoyer au Contrôleur (3), qui à partir des données reçu du modèle (4), va générer une Vue (Page Web) (5) qui sera envoyé au visiteur comme résultat de sa requête (6).

Couche Vue

- C’est la partie de votre code qui s’occupera de la présentation des données à l’utilisateur, elle retourne une vue des données venant du modèle, en d’autres termes c’est elle qui est responsable de produire les interfaces de présentation de votre application à partir des informations qu’elle dispose (page HTML par exemple).
- Cependant, elle n’est pas seulement limitée au HTML (PDF par exemple) ou à la représentation en texte des données, elle peut aussi être utilisée pour offrir une grande variété de formats en fonction de vos besoins.

Couche contrôleur

- C’est la couche chargée de router les informations, elle va décider qui va récupérer l’information et la traiter. Elle gère les requêtes des utilisateurs et retourne une réponse avec l’aide de la couche Modèle et Vue.

La couche Modèle

- C’est la partie de votre code qui exécute la logique métier de votre application. Ceci signifie qu’elle est responsable de récupérer les données, de les convertir selon les concepts de la logique de votre application tels que le traitement, la validation, l’association et tout autre tâche concernant la manipulation des données.
- Elle est également responsable de l’interaction avec la base de données, elle sait en quelque sorte comment se connecter à une base de données et d’exécuter les requêtes (CREATE, READ, UPDATE, DELETE) sur une base de données.

MVC en J2EE

- Avec la plateforme JEE, chaque élément du modèle MVC porte un nom spécifique.
- Le Contrôleur porte le nom de Servlet.
- Le modèle est en général géré par des objets Java ou des JavaBeans. Il peut être amené aussi à communiquer avec les bases de données pour stocker les informations, pour les persister et les garder en mémoire le plus longtemps possible.
- La Vue quant à elle est géré par les pages JSP (Java Server Pages) dans les container Web (Tomcat) ou des JSF (Serveur J2EE)

Introduction à la programmation avec Java

- Les fondements de la programmation
- Les constructions de base d’un programmes
- Tests Logiciels
- Bonne Pratiques de conception d’une application
- Les techniques Objets
  - Les principes généraux de la modélisation et de la programmation Objet.
  - L'abstraction et l'encapsulation : les interfaces.
  - Les différentes formes d'héritage, le polymorphisme.
  - Introduction à la modélisation UML
- Introduction à SVN
- Définition de la structure d’un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Les techniques Objets

Les principes généraux de la modélisation et de la programmation Objet.

- Contenu:
  - Une révision de ce qu’est une classe, attribut, méthode
  - Une révision de la notion de static
  - Présentation de l’héritage
  - Présentation de la notion d’interface
- Objectif:
  - Comprendre l’héritage et la surcharge de méthode

Le Java en Patate

Propriété statique:
- AgeMinimal: 0
- AgeMajeur: 18

Méthode statique:
- Crier(): imprimer « crier »

Propriété:
- Nom
- Age

Méthode:
- DireBonjour : imprime 'Nom'

La classe Humain

Propriété statique:
- AgeMinimal: 0

Méthode statique:
- Crier(): imprimer « crier »

Propriété:
- Nom: Lisa
- Age : 18

Méthode:
- DireBonjour : imprime 'Lisa'

Objet Lisa de class Humain

Propriété statique:
- AgeMinimal: 0

Méthode statique:
- Crier(): imprimer « crier »

Propriété:
- Nom : Pilou
- Age : 20

Méthode:
- DireBonjour : imprime 'Pilou'

Objet Pilou de class Humain

Java

- Pour comprendre Java, il faut voir la notion de « class ».
- Une « class » est une décrit un ensemble de fonctionnalité. Ces fonctionnalité sont appelé « méthode »
- Une instance de class est appelé un objet.
- Un exemple:
  - Nous pouvons décrire un humain. Cette description est appelé class, et nous parlerons de la classe humain.
  - Un etre humain (vous, moi …) est une « instance » d'humain. En l'occurrence le professeur est une instance de la classe humain.

Java

- Une classe a des attributs. Par exemple la classe Humain peut avoir comme attribut le nom de l’humain.
  - On peut dire que moi est une instance de la classe Humain dont la valeur de l’attribut « nom » est « Pierre ».
- La plupart des attributs n’ont de valeur que pour un objet (le nom d’un humain, son age ….), par contre certains attributs ont la même valeur pour tous les objets. On dit que ces attributs sont « static ». (Par exemple l’age minimal d’un humain est 0 quelque soit l’humain. L’age minimal est donc un attribut static de la classe humain.
- Une fonctionnalité (on dit méthode) d’une classe porte la plupart du temps sur un objet

Java

- Par exemple, une fonctionnalité de la classe humain peut permettre d’avoir le nom d’un humain. Cette fonctionnalité peut s’appeler « Donne moi ton nom ».
- Parfois certaine fonctionnalité n’appartiennent a aucun objet et son intrinsèque à la classe. Par exemple, la fonctionnalité permettant d’avoir l’age minimal d’un humain est une fonctionnalité de la classe Humain. Ces fonctionnalité sont dit « static »

Java

- Nous avons donc :
  - Les classes qui sont des descriptions contenant des attributs et des méthodes
  - Les objets qui sont des instances de classes.
  - Les attributs et les méthodes prennent le plus souvent sens pour un objet mais peuvent n’avoir de sens que pour les classes. Il sont dit static.
- Soit une classe A ayant un attribut static B, On écrira A.B
- Soit une classe A ayant une méthode static B sans paramètre, On écrira A.B ()
- Soit une classe A ayant un méthode static B prenant 1 parametre (ici 3) , On écrira A.B(3)
- Soit une classe A ayant un attribut static B ayant lui-même une méthode C, On écrira A.B.C()

Les concepts associés à la programmation objet

- Nous avons déjà vue ce qu’est une classe:
  - Une classe est une définition de structure composé d’attribut.
  - Un objet est l’instanciation d’une classe
  - Les méthodes sont les fonctions des classes
  - Les méthodes/attributs sont static (appartenant à la classe) ou pas (elles nécéssitent alors un objet).
- Nous allons voir:
  - L’héritage ou comment définir qu’une classe est « comme une autre » plus quelque chose
  - Les protections
  - Les interface

Exemple 1

- Un premier exemple avec des Animaux.

Les concepts associés à la programmation objet

- Soit la classe Animal ayant une méthode fait du bruit:
  - Nous voulons implémenté la classe Chien qui est un animal mais qui renvoie ouafouaf dans la méthode faitduBruit
  - Nous voulons aussi implémenté la classe oiseau qui non seulement fait « cuicui » mais qui possède une méthode permettant de savoir combien de temps il peut voler.

```
public class Animal {

      public String faitduBruit()
      {
            return "du bruit";
      }
}
```

Les concepts associés à la programmation objet

- Pour la classe Chien, nous allons dire:
  - Qu'un chien est comme un animal. Il s'agit de la notion d'extension ou d'héritage
  - Qu'un chien lorsqu'il fait du bruit, fait ouafouaf. C'est ce qu'on appel la surcharge

- Remarque: un chien est un animal!!

```
class Chien extends Animal
{
            public String faitduBruit()
            {
                        return "ouafouaf";
            }
}
public static void main(String [] args)
            {
                        Animal animal=new Chien();
                        System.out.println(animal.faitduBruit());
// renvoie ouafouaf
            }
```

Les concepts associés à la programmation objet

- Nous pouvons maintenant crée un objet oiseau qui:
  - Fait cuicui
  - Qui possède un constructeur qui est la méthode d’initialisation des objets

```
class Oiseau extends Animal
{
            public int nbvol;
            public Oiseau(int param_nbvol)
            {
                        nbvol=param_nbvol;
            }
            public String faitduBruit()
            {
                        return "cuicui";
            }
}
```

Les concepts associés à la programmation objet

- En Java, il est possible de n'hériter que d'une seul classe
- On peut redéfinir des méthodes, rajouter des méthodes, rajouter des attributs
- Enfin, il est possible de rendre une méthode « abstraite », qui force les classes « héritantes » à l'implémenter.

```
abstract class Oiseau extends Animal
{
            public int nbvol;
            public Oiseau(int param_nbvol)
            {
                        nbvol=param_nbvol;
            }
            public abstract boolean peutVoler();
            public String faitduBruit()
            {
                        return "cuicui";
            }
}
```

Exemple de méthode abstraite

```
class Poulet extends Oiseau
{
// Ce constructeur est forcé car Oiseau
// a un constructeur prenant un entier
        public Poulet(int param_nbvol) {
                super(param_nbvol);
        }
// Cette méthode est forcé car déclarer abstract dans Oiseau
        public boolean peutVoler() {
                return false;
        }
}
```

Interface

- Enfin, une classe n'ayant pas d'attribut et uniquement des méthodes abstraite est dites une interface.

```
interface ObjetVolant
{
        public void vol();
}
class Poulet extends Oiseau implements ObjetVolant
{

        public Poulet(int param_nbvol) {
                super(param_nbvol);
        }
        public boolean peutVoler() {
                return false;
        }
        public void vol() {

        }
}
```

Exemple 2

- L'idée est de présenter un autre exemple afin de mieux comprendre

Surface

- Les surfaces sont des structures génériques que l'on souhaite pouvoir manipuler (faire des listes de surfaces par exemple).
- Une surface est une structure permet a minima d'avoir l'aire de la surface

```
public interface Surface {

public double computeSurface();
}
```

Un point

- Les surface manipule des point.
  - Un cercle a un centre et un rayon
  - Un quadrilatere possèdent 4 points

```
public class Point {

double x;
double y;
public Point()
{
this.x=0;
this.y=0;
}
public Point(double x, double y) {
super();
this.x = x;
this.y = y;
}
…
```

Un cercle

- Un cercle est une surface, donc on la définit comme tel.
- Cela force a implémenter la méthode surface donnant le résultat d’une surface

```
public class Cercle implements Surface{

Point centre;
double rayon;

public double computeSurface()
{
return Math.PI*rayon*rayon;
}
}
```

Le carré

- Le carré implémente aussi une surface
- Ils possèdent 4 cotés

```
public class Quadrilatere implements Surface{

Point p1;
Point p2;
Point p3;
Point p4;
public double computeSurface()
```

On résume

- L’idée est de revenir sur toutes les notions pour être bien persuadé que l’on a compris.

Les objets

- Un objet:
  - peut être construit
  - Est structuré: il est constitué d'un ensemble d'attribut
  - possède un état: la valeur de ses attributs
  - Peut posséder une interface: l'ensemble des méthodes et propriétés accessibles par les utilisateurs de l'objet
- Dans les langages orientés objet, une classe définit :
  - une façon de construire des objets (constructeur)
  - la structure des objets de la classe (propriétés)
  - le comportement des objets de la classe (méthodes)
  - l'interface des objets de la classe (méth. et prop. non-privées)
  - un type "référence vers des objets de cette classe"

Le mot Class

- mot clé class permet de définir une classe :
  - classCounter{
  - /* Définition des propriétés et des méthodes */}
- On peut ensuite définir une variable de type “référence vers un Counter”:
  - Counter counter;
- Une classe définit également un “moule” pour fabriquer des objets. La création d’une instance s’effectue avec le mot clé new:
  - Counter counter=newCounter();
  - La variable counter contient alors une référence vers une instance de la classe Counter.

Opérateur .

- Les propriétés décrivent la structure de données de la classe : class Counter{intposition;intstep;}
- On accède aux propriétés avec l’opérateur “.” (point) :
  - Counter counter=newCounter();
  - counter.position=12;
  - counter.step=2;
  - counter.position+=counter.step;
- Chaque instance possède son propre état :
  - Counter counter1=newCounter();
  - Counter counter2=new Counter();
  - counter1.position=10;
  - counter2.position=20;
  - if(counter1.position!=counter2.position)System.out.println("différent");

Les méthodes

- Méthodes qui modifient l'objet :
  - Class Counter {
  - int position,step;
  - Void init(int initPos,int s)
  - {position=initPos;step=s;}
  - Void count(){position+=step;}}
- On invoque les méthodes avec l'opérateur "." (point) :
  - Counter counter=newCounter();
  - counter.init(4,8+12);
  - counter.count();
  - counter.count();
  - System.out.println(counter.position);

Constructeur

- Si aucun constructeur n’est défini, la classe a un constructeur par défaut :
  - public class Counter { int position=10,step=1;}
- Ce code est équivalent à celui-ci :
  - Public class Counter { int position,step; publicCounter(){position=10;step=1};
- Le mot-clé this fait référence à l’instance en construction ou à l’instance qui a permis d’invoquer la méthode en cours d’exécution :
  - Class Counter {intposition,step; Counter(intposition,intstep)
  - {this.position=position;this.step=step;}
  - Void count(){intposition=this.position;position+=step;this.position=position;}}

Constructeur

- Le mot-clé this permet également d’appeler un constructeur dans un constructeur de façon à ne dupliquer du code :

```
class Counter {
int position, step ;
Counter(int position, int step) {
this.position = position ; this.step = step ;
}
Counter(int position) {
this(position, 1);
}
/* Autres méthodes. */
}
```

Surcharge de méthode

- Dans une classe, deux méthodes peuvent avoir le même nom. Java doit pouvoir décider de la méthode à exécuter en fonction des paramètres :

```
class Value {
int value = 0 ;
void set() { value = 0 ; }
void set(int value) { this.value = value ; }
void set(double vallue) { this.value = (int)value ; }
}
```

- Exemple
  - Value value = **new** Value();
  - value.set(); // Première méthode
  - value.set(2); // Deuxième méthode
  - value.set(2.5); // Troisième méthode

Propriétés et méthodes statiques

- Les propriétés et des méthodes statiques sont directement associées à la classe et non aux instances de la classe :

```
class Counter {
static int step ;
int position ;
Counter(int position) { this.position = position ; }
static void setStep(int step) {
Counter.step = step ;
}
void count() { position += step ; }
}
```

Données et méthodes statiques

- Comme les propriétés et méthodes statiques sont associées à la classe, il n’est pas nécessaire de posséder une instance pour les utiliser :

```
public class Test {
public static void main(String[] arg) {
Counter.setStep(3);
Counter counter1 = new Counter(2);
Counter counter2 = new Counter(3);
counter1.count(); counter2.count();
System.out.println(counter1.position); // ! 5
System.out.println(counter2.position); // ! 6
Counter.setStep(4); counter1.count(); counter2.count();
System.out.println(counter1.position); // ! 9
System.out.println(counter2.position); // ! 10
}
}
```

Données et méthodes statiques

Une méthode statique ne peut utiliser que :

- des propriétés statiques à la classe ;
- des méthodes statiques à la classe ;

afin de garantir par transitivité l'utilisation exclusive de données statiques.

L'utilisation de this n'a aucun sens dans une méthode statique :

```
class Counter {
/* ... */
static void setStep(int step) {
Counter.step = step ;
}
/* ... */
}
```

On résume

- Abstraction et Interface

Abstraction

- Supposons que des classes implémentent un service de façons différentes :

```
class SimplePrinter {
void print(String document) {
System.out.println(document);
}
}
class BracePrinter {
void print(String document) {
System.out.println("{"+document+"}");
}
}
```

Toutes les instances de ces classes possèdent la méthode print

Interface

- Description d’une interface en Java :

```
public interface Printer {
/**
* Affiche la chaine de caracteres document.
* @param document la chaine a afficher.
*/
public void print(String document);
}
```

Une interface :

- peut contenir une ou plusieurs méthodes
- est un contrat

Interface

implements permet d’indiquer qu’une classe implémente une interface :

```
class SimplePrinter implements Printer {
void print(String document) {
System.out.println(document);
}
}
class BracePrinter implements Printer {
void print(String document) {
System.out.println("{"+document+"}");
}
}
```

Java vérifie que toutes les méthodes de l’interface sont implémentées

Interface

- Déclaration d'une variable de type référence vers une instance d'une classe qui implémente l'interface Printer :
  Printer printer ;
  Il n'est pas possible d'instancier une interface :
  Printer printer = **new** Printer(); *// interdit !*
  printer.print("toto"); *// que faire ici ?*

- Il est possible d'instancier une classe qui implémente une interface :
  SimplePrinter simplePrinter = **new** SimplePrinter();
  et de mettre la référence dans une variable de type Printer :
  Printer printer1 = simplePrinter ; *// upcasting*
  Printer printer2 = **new** BracePrinter();
  Printer string = **new** String("c") *// interdit !*

Interface

- Une interface est un ensemble de signatures de méthodes.
- Une classe peut implémenter une interface : elle doit préciser le comportement de chacune des méthodes de l’interface.
- Il est possible de déclarer une variable pouvant contenir des références vers des instances de classes qui implémentent l’interface.
- Java vérifie à la compilation que toutes les affectations et les appels de méthodes sont corrects.
- Le choix du code qui va être exécuté est décidé à l’exécution (en fonction de l’instance pointée par la référence)

Interface Multiple

interface Printable {

public void print();

}

La méthode print permet d’afficher l’objet.

interface Stack {

public void push(**int** value);

public int pop();

}

La méthode push permet d’empiler un entier.

La méthode pop dépile et retourne l’entier en haut de la pile

Interface Multiple

Implémentation des deux interfaces précédentes :

```
public class PrintableStack implements Stack, Printable {
private int[] array ; private int size ;
public PrintableStack(int capacity) {
array = new int[capacity]; size = 0 ;
}
public void push(int v) { array[size] = v ; size++; }
public int pop() { size-- ; return array[size]; }
public void print() {
for (int i = 0 ; i < size ; i++)
System.out.print(array[i]+" ");
System.out.println();
}
}
```

Extends

- Plus généralement, l’extension permet de créer une classe en :
  - conservant les services (propriétés et méthodes) d’une autre classe ;
  - ajoutant de nouveaux services (propriétés et méthodes) ;
  - redéfinissant certains services (méthodes).
- En Java :
  - On utilise le mot-clé extends pour étendre une classe ;
  - Une classe ne peut étendre qu’une seule classe.

Super

Supposons que nous ayons la classe Point suivante :

```
public class Point {
public int x, y ;
public void setPosition(int x, int y) { this.x = x ; this.y = y ; }
public void clear() { setPosition(0, 0); }
}
```

Le mot-clé super permet d'utiliser la méthode clear de Point :

```
public class Pixel extends Point {
public int r, g, b ;
public void setColor(int r, int g, int b)
{ this.r = r ; this.g = g ; this.b = b ; }
public void clear() { super.clear(); setColor(0, 0, 0); }
}
```

Super

Supposons que nous ayons la classe Point suivante :

public class Point {

public int x, y ;

public void setPosition(**int** x, **int** y) { **this**.x = x ; **this**.y = y ; }

}

Si la méthode n'a pas été redéfinie, le mot clé super est inutile :

public class Pixel extends Point {

public int r, g, b ;

public void setColor(**int** r, **int** g, **int** b)

{ **this**.r = r ; **this**.g = g ; **this**.b = b ; }

public void clear() { */*super.*/*setPosition(0, 0); setColor(0, 0, 0); }

}

Constructeur et Super

Supposons que nous ayons la classe Point suivante :

```
public class Point {
public int x, y ;
public Point(int x, int y) { this.x = x ; this.y = y ; }
}
```

La classe Point n'a pas de constructeur sans paramètre, il faut donc indiquer comment initialiser la partie de Pixel issue de la classe Point :

```
public class Pixel extends Point {
public int r, g, b ;
public Pixel(int x, int y, int r, int g, int b) {
super(x, y); // appel du constructeur de Point
this.r = r ; this.g = g ; this.b = b ;
}
}
```

Extension d’Interface

Supposons que nous ayons l’interface suivante :

```
public interface List {
public int size();
public void get(int index);
}
```

En Java, il est également possible d’étendre une interface :

```
public interface ModifiableList extends List {
public void add(int value);
public void remove(int index);
}
```

Une classe qui implémente l’interface ModifiableList doit implémenter les méthodes size, get, add et remove.

On résume

- Les Enumérations

Enum

Il est possible de définir des énumérations :

```
enum Suit {
SPADES,
HEARTS,
DIAMONDS,
CLUBS ;
}
```

Une énumération est une classe avec des éléments prédéfinis et statiques :

```
Suit suit = Suit.SPADES ;
/* ... */
if (suit == Suit.SPADES) { /* .... */ }
```

Enum

Définition de champs, de méthodes et d’un constructeur :

```
enum Suit {
SPADES("Pique", "Pi"),
HEARTS("Coeur", "Co"),
DIAMONDS("Carreau", "Ca"),
CLUBS("Trèfle", "Tr");
private final String name ;
private final String symbol ;
Suit(String name, String symbol) {
this.name = name ;
this.symbol = symbol ;
}
public String name() { return name ; }
public String symbol() { return symbol ; }
}
```

Enum

Un exemple d'utilisation de l'énumération précédente :

```
public static void main(String[] args) {
for (Suit suit : Suit.values())
System.out.printf("Le symbole de %s est %s",
suit.name(),
suit.symbol());
}
```

Un autre exemple :

```
public static void main(String[] args) {
for (Suit suit : Suit.values())
System.out.printf("La position de %s est %d",
suit.name(),
suit.ordinal());
}
```

On résume

- Les paramétriques

Problématique

Supposons que nous ayons la classe suivante :

```
public class Stack {
private Object[] stack = new Object[100];
private int size = 0 ;
public void push(Object object) {
stack[size] = object ; size++;
}
public Object pop() {
size-- ; Object object = stack[size];
stack[size]=null ; // Pour le Garbage Collector.
return object ;
}
}
```

Problématique

Nous rencontrons le problème suivant :

```
Stack stack = new Stack();
String string = "truc" ;
stack.push(string);
string = (String)stack.pop(); // Transtypage obligatoire !
```

Nous avons également le problème suivant :

```
Stack stack = new Stack();
Integer integer = new Integer(2);
stack.push(integer);
String string = (String)p.pop(); // Erreur à l'exécution
```

Class Paramétré

Par conséquent, on souhaiterait pouvoir préciser le type des éléments :

```
Stack<String> stack = new Stack<String>();
```

```
String string = "truc" ;
stack.push(string); // Le paramètre doit être un String.
string = p.pop(); // La méthode retourne un String.
```

Java nous permet de définir une classe Stack qui prend en paramètre un type. Ce type paramétré va pouvoir être utilisé dans les signatures des méthodes et lors de la définition des champs de la classe.

Lors de la compilation, Java va utiliser le type paramétré pour effectuer :

- des vérifications de type ;
- des transtypages automatiques ;
- des opérations d’emballage ou de déballage de valeurs

Class Paramétré

La nouvelle version de la classe Stack :

```
public class Stack<T> {
private Object[] stack = new Object[100];
private int size = 0 ;
public void push(T t) { stack[size] = t ; size++; }
public T pop() {
size-- ;
T t = (T)stack[size];
stack[size] = null ;
return t ;
}
}
```

Class Parametre

Les types simples ne sont pas des classes :

Stack<**int**> stack = **new** Stack<**int**>(); *// interdit !*

Dans ce cas, on doit utiliser la classe d'emballage Integer :

Stack<Integer> stack = **new** Stack<Integer>();

int intValue = 2 ;

Integer integer = **new** Integer(intValue);

// ,! emballage du int dans un Integer.

stack.push(integer);

integer = stack.pop();

intValue = integer.intValue();

// ,! déballage du int présent dans le Integer.

Rappel

short	entier	16 bits	-32768 à 32767	0
int	entier	32 bits	-2^{31} à $2^{31} - 1$	0
long	entier	64 bits	-2^{63} à $2^{63} - 1$	0
float	flotant	32 bits	0.0	
double	flotant	64 bits	0.0	
char	caractère	16 bits	caractères Unicode	\u0000
boolean	boolean	1 bit	false ou true	false

Wrapper de type natif

Les classes d’emballage :

- Byte -> public Byte(byte b)
- Short -> public Short(short s)
- Integer -> public Integer(int i)
- Long -> public Long(long l)
- Float -> public Float(float f)
- Double -> public Double(double d)
- Boolean -> public Boolean(boolean b)
- Character -> public Character(char c)

Autoboxing

Depuis Java 5, il existe l'emballage et le déballage automatique :

```
Stack<Integer> stack = new Stack<Integer>();
int intValue = 2 ;
stack.push(intValue);
// ,! emballage automatique du int dans un Integer.
intValue = stack.pop();
// ,! déballage automatique du int.
```

Attention : il est important de noter que des allocations sont effectuées lors des emballages sans que des new soient présents dans le code

Collections

Des interfaces :

- Collection<V> : Groupe d’éléments
- List<V> : Liste d’éléments ordonnés et accessibles via leur indice
- Set<V> : Ensemble d’éléments uniques
- Queue<V> : Une file d’éléments (FIFO)
- Deque<V> : Une file à deux bouts (FIFO-LIFO)
- Map<K,V> : Ensemble de couples clé-valeur.

Il est préférable d’utiliser les interfaces pour typer les variables :

List<Integer> list = **new** ArrayList<Integer>();

/ code qui utilise list. */*

car on peut changer la structure de données facilement :

List<Integer> list = **new** LinkedList<Integer>();

/ code qui utilise list et qui n'a pas à être modifié. */*

Collections

Implémentations de List<V> :

- ArrayList<V> : Tableau dont la taille varie dynamiquement.
- LinkedList<V> : Liste chaînée.
- Stack<V> : Pile (mais une implémentation de Deque est préférable).

Implémentations de Map<K,V> :

- HashMap : Table de hachage.
- LinkedHashMap : Table de hachage + listes chainées.
- TreeMap : Arbre rouge-noir (éléments comparables)

On résume

- Les Exceptions

Exceptions

Un programme peut être confronté à une condition exceptionnelle (ou exception) durant son exécution.

Une exception est une situation qui empêche l'exécution normale du programme (elle ne doit pas être considérée comme un bug).

Quelques exemples de situations exceptionnelles :

- un fichier nécessaire à l'exécution du programme n'existe pas ;
- division par zéro ;
- débordement dans un tableau ;
- etc.

Exceptions

Java propose un mécanisme de gestion des exceptions afin de distinguer l'exécution normale du traitement de celles-ci afin d'en faciliter leur gestion.

En Java, une exception est concrétise par une instance d'une classe qui étend la classe Exception.

Pour lever (déclencher) une exception, on utilise le mot-clé throw :

if (problem) **throw new** MyException("error");

Pour capturer une exception, on utilise la syntaxe try/catch :

try { */* Problème possible */* }

catch (MyException e) { */* traiter l'exception. */* }

Definir une exception

Il suffit d'étendre la classe Exception (ou une classe qui l'étend) :

```
public class MyException extends Exception {
private int number ;
public MyException(int number) {
this.number = number ;
}
public String getMessage() {
return "Error "+number ;
}
}
```

Try catch

Pour capturer une exception, on utilise la syntaxe try/catch :

```
public static void test(int value) {
System.out.print("A ");
try {
System.out.println("B ");
if (value > 12) throw new MyException(value);
System.out.print("C ");
} catch (MyException e) { System.out.println(e); }
System.out.println("D");
}
```

test(11) :

A B

C D

test(13) :

A B

MyException: Error 13

D

Méthode et Exception

Une méthode doit préciser dans sa signature toutes les exceptions qu'elle peut générer et qu'elle n'a pas traitées avec un bloc try/catch :

```
public class Test {
public static void runTestValue(int value) throws MyException {
testValue(value);
}
public static void testValue(int value) throws MyException {
if (value>12) throw new MyException(i);
}
public static void main(String args[]) {
try { runTestValue(13); }
catch (MyException e) { e.printStackTrace(); }
}
}
```

RuntimeException

Une méthode doit indiquer toutes les exceptions qu'elle peut générer sauf si l'exception étend la classe RuntimeException Bien évidemment, la classe RuntimeException étend Exception.

Quelques classes Java qui étendent RuntimeException :

- ArithmeticException
- ClassCastException
- IllegalArgumentException
- IndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException

- Notez que ces exceptions s'apparentent le plus souvent à des bugs.

Multi Catch

Il est possible de capturer une exception en fonction de son type :

```
public static int divide(Integer a, Integer b) {
try { return a/b ; }
catch (ArithmeticException exception) {
exception.printStackTrace();
return Integer.MAX_VALUE ;
} catch (NullPointerException exception) {
exception.printStackTrace();
return 0 ;
}
}
```

divide(12,0) :

```
java.lang.ArithmeticException: / by zero
at Test.diviser(Test.java:17)
at Test.main(Test.java:28)
```

divide(null,12) :

```
java.lang.NullPointerException
at Test.diviser(Test.java:17)
at Test.main(Test.java:28)
```

finally

Le bloc associé au mot-clé finally est toujours exécuté :

```
public static void readFile(String fileName) {
try {
FileReader fileReader = new FileReader(fileName);
/* peut déclencher une FileNotFoundException. */
try {
int character = fileReader.read(); /* IOException ? */
while (character !=-1) {
System.out.println(character);
character = fileReader.read(); /* IOException ? */
}
} finally { fileReader.close(); /* à faire dans tous les cas. */ }
} catch (IOException exception) { exception.printStackTrace(); }
}
```

FileNotFoundException étend IOException donc elle est capturée.

Introduction à la programmation objet

La modélisation objet à partir des exigences fonctionnelles

- Contenu:
  - Une approche de UML comme outils à la conception

- Objectif:
  - Connaitre la terminologie UML
  - Voir comment se l’approprier

Problématique de modélisation

- Lorsqu'on débute, il est difficile de voir pour un projet complexe comment prendre la situation.
- Il est possible d'utiliser quelques diagrammes issue des travaux UML afin de faciliter:
  - La discussion avec les utilisateurs (que doit faire le produit)
  - La discussion avec les développeurs (comment le produit est fait).
  - La mise en place de la documentation
- En pratique, sauf cas spécifique, les diagrammes UML sont utilisé comme aide à la conception.

Diagramme simple

- Il y a trois diagrammes utiles et « simples » pour débuter:
  - Le diagramme de cas d'usage qui doit montrer ce que doit faire un utilisateur avec le produit
  - Le diagramme de classe plus technique qui montrent les différentes classes, methodes ainsi que leurs interractions
  - Le diagramme de séquence qui tente de rassembler « lier » le cas d'usage avec le diagramme de classe
- Un exemple classique est celui du distributeur de banque:
  - Il est possible de rentrer un code personnel
  - De choisir un montant d'argent à récupérer (20, 40, 50 et 100 euros)
  - De récupérer sa carte soir après avoir récupérer l'argent soit avant

Diagramme d’usage

- On represente le système (un carré)
- Les usages (des ronds)
- Des types d’usagers (les humains)
- On fait un lien entre les usages et les usagers

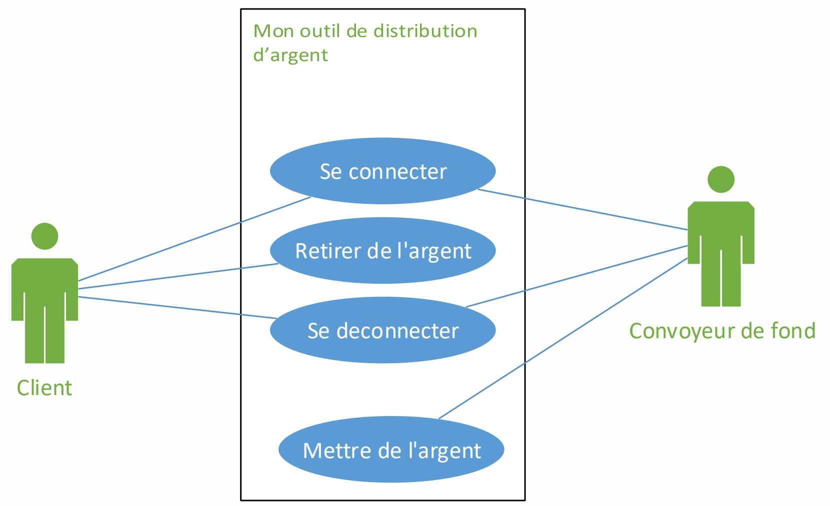
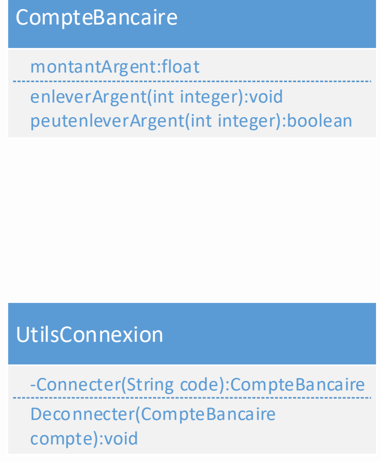
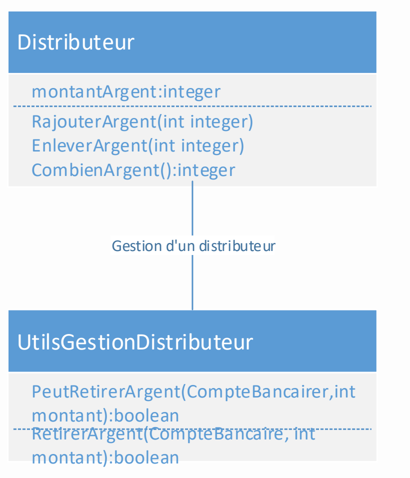


Diagramme de classe

- On essaye de représenter les classes java comme des carré avec leur méthode.
- On représente un trait entre deux classes pour associer les classes.





Un diagramme de classe plus complexe

- Les flèches représente l’héritage entre classe
- Les flèches pointillé les implémentation d’interface

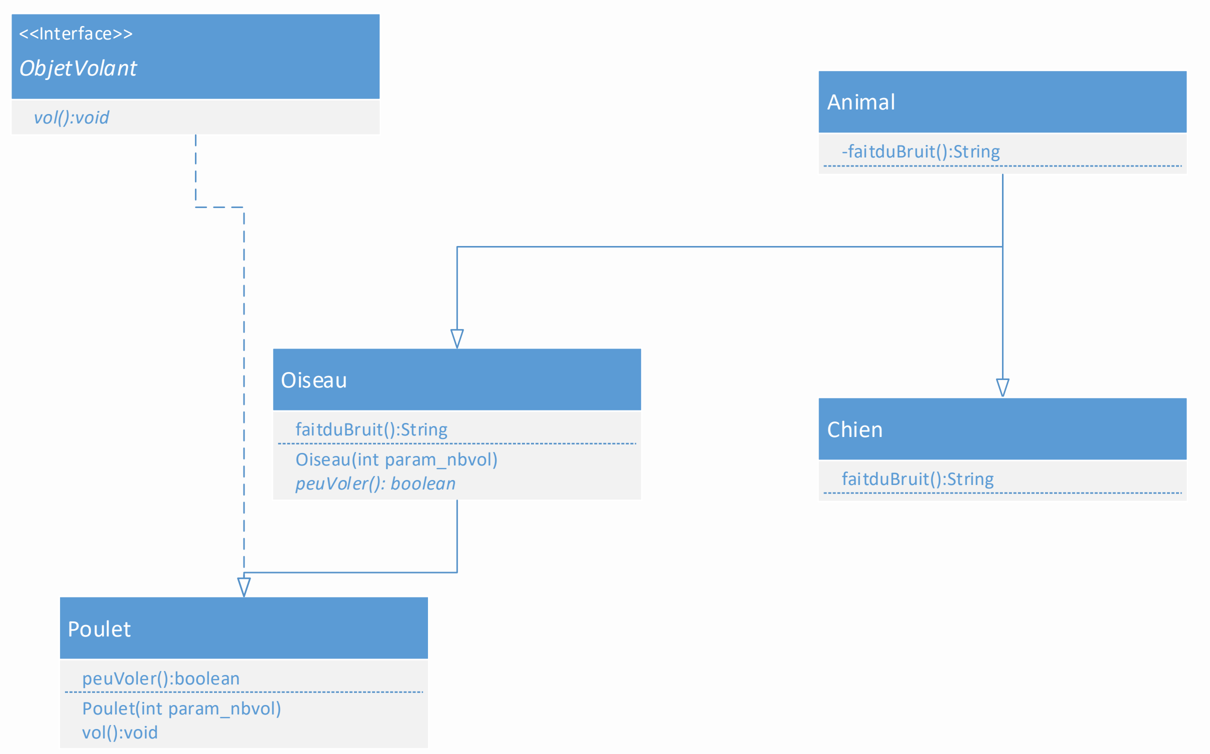
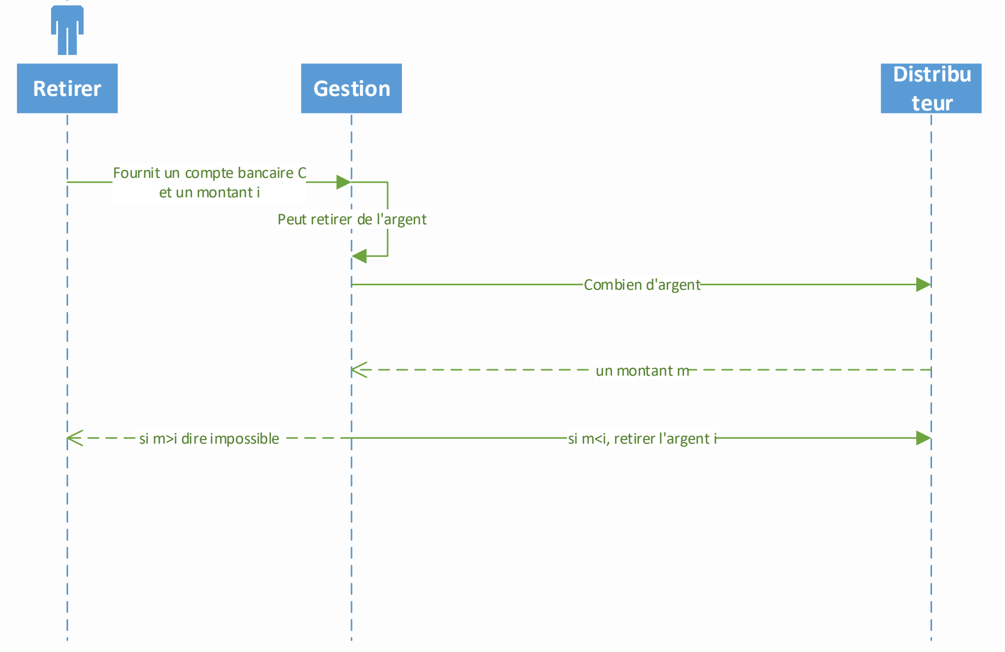


Diagramme de séquence

- Pour chaque usage, on tente de mettre en relation les différentes classses.
- La première barre est soit un usage soit une classe
- La temporalité va de gauche à droite



Introduction à la programmation objet

Introduction aux bonnes pratiques d'organisation de conception et d'organisation d'un programme.

- Contenu:
  - Comment organiser un programme
  - Quelques éléments de « syntaxes »
- Objectif:
  - Architecturer un programme

Comment architecturer un programme

- Un programme Java contient rapidement beaucoup de fichiers que cela soit des sources, des binaires, des librairies…

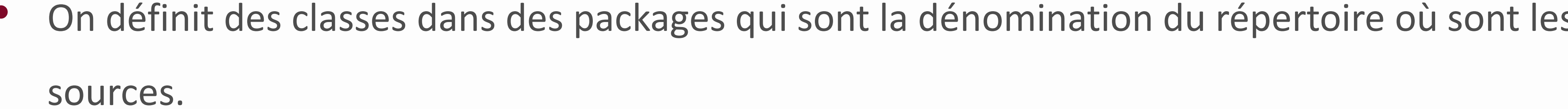
- L'idée général est de voir comment organiser pour pouvoir se retrouver.

- De se donner quelques règles d'écriture assez commune

Organisation des répertoires

- En général on trouve comme répertoire:
  - Src contenant les sources du programmes
  - Bin ou Jar contenant un fichier Jar (un zip) des binaires compilés
  - Optionnellement un répertoire class contenant les sources compilés mais non agrégé
  - Un répertoire doc contenant les fichiers de documentation
  - Un répertoire thirdparties contenant les librairies annexes utilisés.

Organisation d’une classe



- On définit des classes dans des packages qui sont la dénomination du répertoire où sont les sources.
  - Par exemple les fichiers qui sont dans le répertoire src\com\neuresys\utils sont définie dans un package com.neuresys.utils.
- On ne définit pas de classes à la racine du repertoire src mais toujours un package
- On importe uniquement les classes dont on a besoin.

Nomination des classes

- Une interface définit en fait un concept (par exemple ObjetVolant) et doit donc porter un nom de concept
- Les classes sont des implémentations dites « concrète » qui en général:
  - Rappel le nom de l'interface si il y a une implémentation (par exemple PouletObjetVolant)
  - Porte parfois le suffice Impl pour montrer qu'il s'agit d'une implémentation assez générique d'une interface (ce qui donne ObjetVolantImpl)
  - Porte en préfixe le mot Abstract si une des méthodes de la classes est abstraites

Nomination des méthodes

- Si le but d'une méthode est uniquement de renvoyé une propriété, cette méthode est préfixé par « get »
  - Pour les propriétés booléenes, une tel méthode est préfixé par is
- Si le but d'une méthode est de mettre a jour une propriété, cette méthode est préfixé par set
- Enfin les noms de méthodes essaye d'être explicite sur ce que font ces méthodes

Introduction à la programmation avec Java

- Les fondements de la programmation
- Les constructions de base d'un programmes
- Tests Logiciels
- Bonne Pratiques de conception d'une application
- Les techniques Objets
- Introduction à SVN
  - Les concepts généraux liés à la gestion de versions.
  - Les concepts SVN : dépôt, projets, révisions, tronc, branches et tags.
  - Les principales opérations offertes au développeur. La gestion des conflits.
  - La gestion des branches. Les perspectives SVN proposées par les plug-ins Eclipse
- Définition de la structure d'un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Introduction à SVN

Les concepts généraux liés à la gestion de versions

- Besoin d’un logiciel de gestion de version
- Quels sont les grand type de logiciel?

Les concepts généraux liés à la gestion de versions

- Le besoin d'organiser et de conserver les versions d'un code ont apparue assez tôt dans le monde de l'informatique (86 avec CVS ou RCS).
- Le but est de permettre :
  - De sauvegarder ses sources sur un serveur stable et non sur un poste de travail
  - Permettre de revenir en arrière sur des modifications (en cas de régressions par exemple)
  - Permettre à plusieurs personnes de travailler en même temps.
- L'idée est de sauvegarder les versions (ou les différences de versions) sur un serveur. Chaque version est appelé « révision » et possède a minima une date/nom de l'auteur

Modèle de gestion de version

- Avec les logiciels de gestion de versions centralisée, comme CVS et Subversion (SVN), il n'existe qu'un seul dépôt des versions qui fait référence.

- Cela simplifie la gestion des versions mais est contraignant pour certains usages comme le travail sans connexion au réseau, ou tout simplement lorsque l'on travaille sur des branches expérimentales ou contestées.

Modèle de gestion de version

- La gestion de versions décentralisée consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs. De fait, il existe plusieurs dépôts pour un même logiciel. Ce système est très utilisé par les logiciels libres.
- Par exemple, GNU Arch, Git et Mercurial sont des logiciels de gestion de versions décentralisée.
- Avantages de la gestion décentralisée :
  - permet de ne pas être dépendant d'une seule machine comme point de défaillance ;
  - permet aux contributeurs de travailler sans être connecté au gestionnaire de version ;
  - la plupart des opérations sont plus rapides car réalisées en local (sans accès réseau) ;
  - permet le travail privé pour réaliser des brouillons sans devoir publier ses modifications et gêner les autres contributeurs ;
  - permet toutefois de garder un dépôt de référence contenant les versions livrées d'un projet.
- Inconvénients :
  - cloner un dépôt est plus long que récupérer une version pour une gestion de version décentralisée car tout l'historique est copié (ce qui est toutefois un avantage par la suite) ;
  - il n'y a pas de système de lock (ce qui peut poser des problèmes pour des données binaires qui ne se fusionnent pas).

Modèle de gestion de version

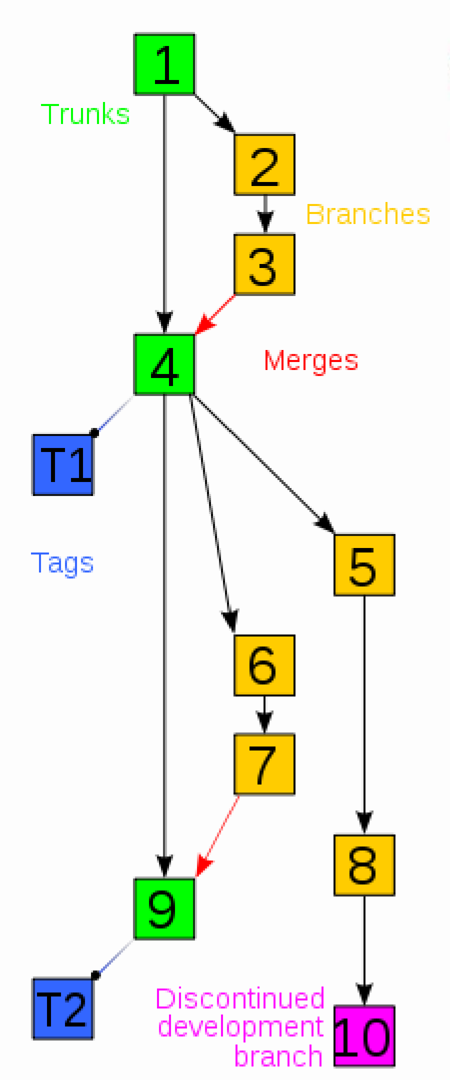
- Les fichiers versionnés sont mis à dispositions sur un dépôt, c'est-à-dire un espace de stockage public géré par un logiciel de gestion de versions.

- Pour pouvoir effectuer des modifications, le développeur doit d'abord faire une copie locale des fichiers qu'il souhaite modifier, ou de tout le dépôt. Selon les systèmes de gestion de version, certains fichiers peuvent être verrouillés ou protégés en écriture pour tout le monde, ou pour certaines personnes.

- Le développeur fait ses modifications et effectue ses premiers tests localement, indépendamment des modifications faites sur le dépôt du fait du travail simultané d'autres développeurs. Il doit ensuite faire un commit (une soumission), c'est-à-dire soumettre ses modifications, afin qu'elles soient enregistrées sur le dépôt. C'est là que peuvent apparaître des conflits entre ce que le développeur souhaite soumettre et les modifications effectuées depuis la dernière copie locale effectuée. Ces conflits doivent être résolus (merge) pour que le patch soit accepté sur le dépôt.

Modèle de gestion de version

- Les concepts sont assez simple. Principalement, il y a une ligne de code principale (Trunks) .
- Des clones de Trunks permettent de faire des développement spécifiques (des branches).
- Généralement une partie des modifications d’une branche vient dans le trunk, il s’agit d’un merge
- Certaines version du trunk sont marqué via un Tag



Modèle de gestion de version

- Lorsque des modifications divergentes interviennent hors conflit, il se crée des branches. Le fait de vouloir rassembler deux branches est une fusion de branches.

- Les branches sont utilisées pour permettre :
  - la maintenance d'anciennes versions du logiciel (sur les branches) tout en continuant le développement des futures versions (sur le tronc) ;
  - le développement parallèle de plusieurs fonctionnalités volumineuses sans bloquer le travail quotidien sur les autres fonctionnalités.

- Les correctifs de la dernière version doivent être faits sur le trunk.

Modèle de gestion de version

- Dans le cas d'un développement en équipe, surtout si elles sont réparties dans le monde entier, il est nécessaire de partager une base commune de travail, et c'est tout l'intérêt des systèmes de gestion de version. Mais, il faut aussi veiller à coordonner les équipes de développement grâce à des outils de communication, un logiciel de suivi de problèmes, un générateur de documentation et/ou un logiciel de gestion de projets.

- Il n'est pas rare que certaines modifications soient contradictoires (par exemple lorsque deux personnes ont apporté des modifications différentes à la même partie d'un fichier). On parle alors de conflit de modifications car le logiciel de gestion de versions n'est pas en mesure de savoir laquelle des deux modifications il faut appliquer.

Introduction à SVN

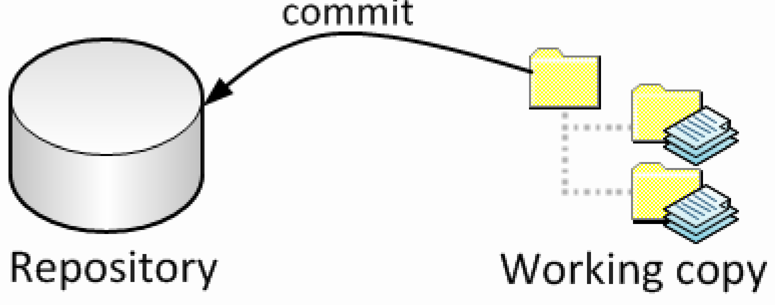
Concept spécifique à SVN

- L’idée est de faire un focus sur SVN

Notions dans SVN

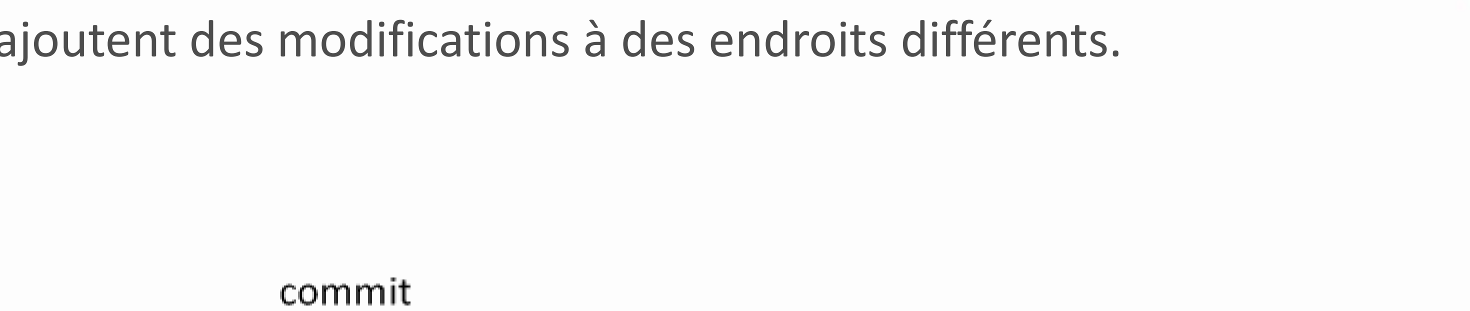
- Dans le cas simple (SVN) : il existe un *référentiel* (repository) et une *copie de travail (working copy)* . Le référentiel est une base de données des versions précédentes; vous travaillez sur la copie de travail.

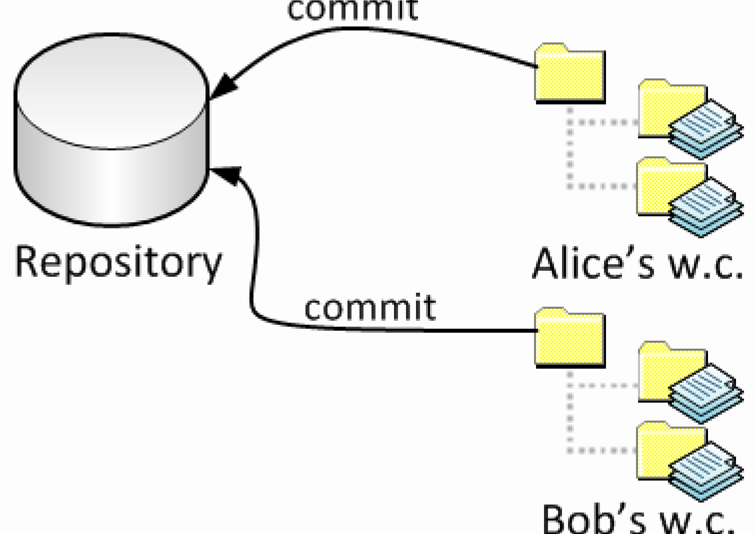
- Vous pouvez **valider** (commit) les modifications que vous avez apportées à la copie de travail dans le référentiel. Cela enregistre les modifications apportées à la copie de travail en tant que nouvelle révision



Multi commit

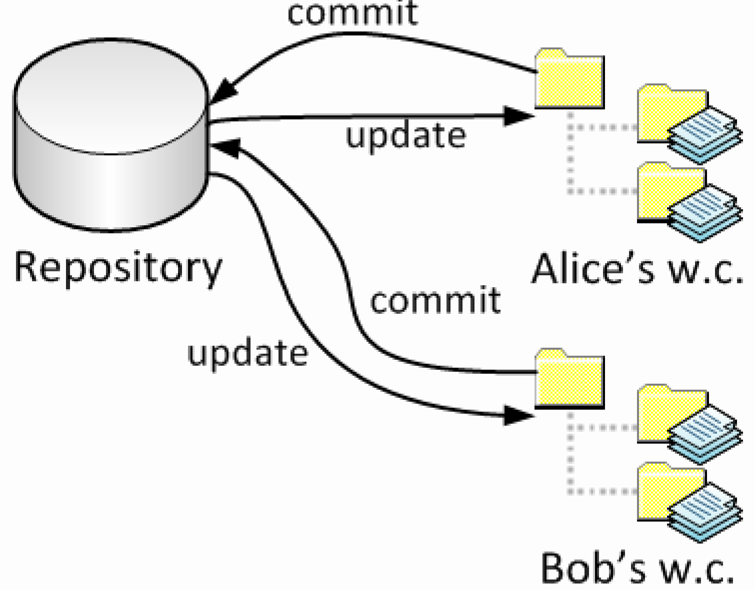
- Qu'en est-il de plusieurs personnes? Dans un certain sens, le cas le plus simple il n’y a pas de problème, les personnes rajoutent des modifications à des endroits différents.





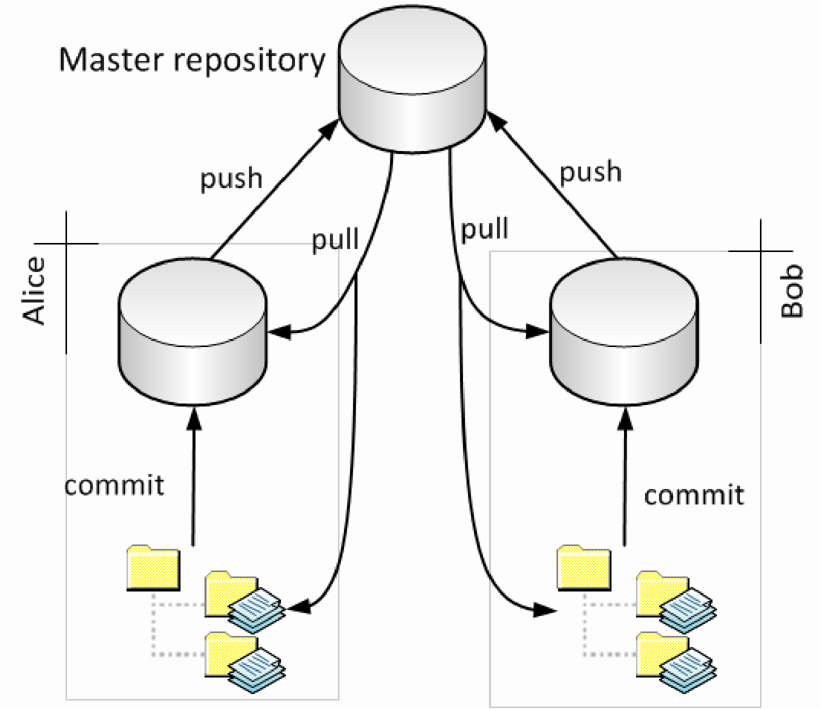
Multi commit

- Maintenant, qu'advient-il de Bob quand Alice commit? Immédiatement rien. Mais comme il y a plus d'une personne, Bob n'a plus la version la plus récente dans sa copie de travail. À un moment donné, il demandera à Subversion de se procurer la version la plus récente: il s’agit d’une *mise à jour* (update):



Modèle en version distribué (git par exemple)

- Chaque utilisateur a son propre repository avec un repository principal
- Dans ce modèle on commit dans son repository et on pousse les modifications de son repository vers le repository principal.



Problématique de la fusion : le lock

- Les problèmes arrivent lorsque deux utilisateurs souhaitent modifier le même fichier.
- Ils existent deux modèles dans SVN, le lock et la fusion.
- Dans un tel système, le dépôt ne permet qu'à une seule personne à la fois de modifier un fichier. Tout d'abord, Alice doit *locker* le fichier avant qu'il ne puisse commencer à y faire des changements.
- Si Alice a verrouillé un fichier, alors Bob ne peut pas le modifier. Si il essaye de verrouiller le fichier, le dépôt refusera la requête. Tout qu'il peut faire est lire le fichier et attendre que Bob finisse ses changements et relâche le verrou.

Problématique de la fusion : la fusion

- L’autre solution est de laisser Alice et Bob édite les fichiers et de gérer les problèmes à la fin.
- Si Bob et Alice ont chacun une copie de travail du même projet, extrait du dépôt.
- Ils travaillent en même temps et modifient localement le même fichier A.
- Alice sauvegarde ses changements dans le dépôt d'abord.
- Quand Bob essaie de sauvegarder ses changements, le dépôt l'informe que son fichier A est périmé. Autrement dit, dans le dépôt, ce fichier A a été modifié depuis qu'il a été extrait.
- Donc Bob doit fusionner les fichiers. Si les changements d’Alice et Bob ne se chevauchent pas, il peut pousser sa version dans le dépôt

Problématique de la fusion

- Il est possible que Alice et Bob est fait les changements au même endroit.
- Dans ce cas, Bob doit fusionner les derniers changements du dépôt vers sa copie de travail, sa copie du fichier A est marquée d'une façon ou d'une autre comme étant dans un état de conflit : il sera capable de voir les deux jeux de changements en conflit et choisira manuellement entre eux

Vocabulaires SVN

- **Repository** ou **dépôt** : C'est simplement le répertoire, sur le serveur **Subversion**, qui va accueillir les fichiers du projet ainsi que les données sur les versions.
- **Checkout** : C'est l'opération qui consiste à obtenir une copie locale de la dernière version d'un projet sur le repository.
- **Commit** : C'est l'opération qui consiste à envoyer les changements d'un fichier sur le serveur **Subversion**.
- **Update** : C'est l'opération qui permet de mettre à jour la version locale d'un fichier avec la dernière version du serveur.
- **Merge** : C'est l'opération qui permet de mettre à jour un fichier local qui a été modifié avec les mises à jour du repository. Cette opération est déclenchée quand le fichier a été modifié en local et qu'il y a également des modifications sur la version du dépôt.
- **Branche** : C'est en fait une sous-version (dérivation) de la version principale. Si on a une seule version tout au long du développement, on n'aura pas de branches, mais si on doit faire une sous-version d'une version spécifique, on devra alors créer une nouvelle branche qui évoluera alors indépendamment de la branche principale
- **Tag** : C'est une capture du repository à un moment précis. Par exemple, pour la beta2 de notre projet, on va faire un tag "beta2" et y copier le contenu de la branche sur laquelle on travaille. Un tag n'est pas destiné à évoluer, c'est une version figée.

Introduction à SVN

Gestion pratique de SVN

- La partie théorique est bien comprise, mais en pratique on fait comment?

Création d'un repository

- Il est possible de créer un dépôt via la commande
  - svnadmin create <nom du repo>

```
sudo svnadmin create /var/lib/svn/myetnicrepo
Ls /var/lib/svn/myetnicrepo
drwxr-xr-x 2 root     root     4096 Oct 11 09:47 conf
drwxr-sr-x 6 root     root     4096 Oct 11 09:47 db
-r--r--r-- 1 root     root        2 Oct 11 09:47 format
drwxr-xr-x 2 root     root     4096 Oct 11 09:47 hooks
drwxr-xr-x 2 root     root     4096 Oct 11 09:47 locks
-rw-r--r-- 1 root     root      246 Oct 11 09:47 README.txt
```

Création d’utilisateur

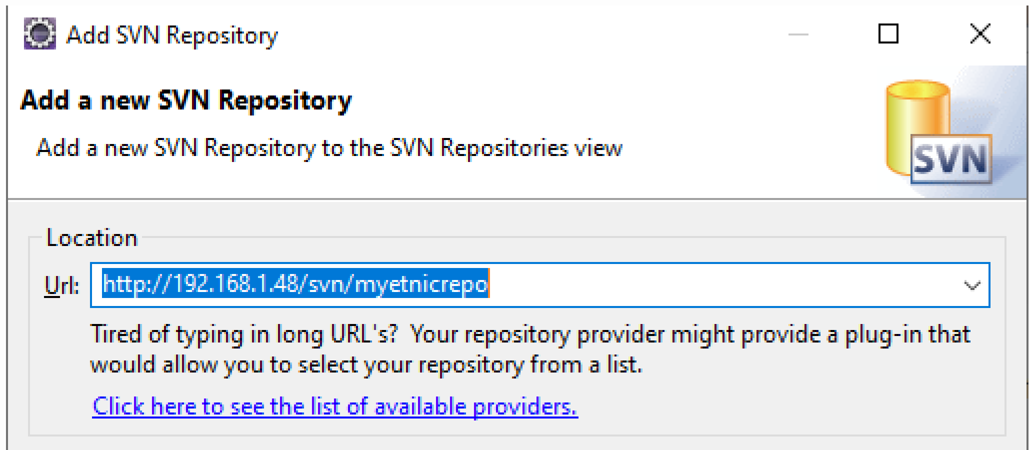
- De façon classique SVN est héberger par un serveur web et expose ses fichiers via webdav.
- La création des utilisateurs se fait via apache

```
sudo htpasswd -m /etc/apache2/dav_svn.passwd alice
New password:
Re-type new password:
Adding password for user alice
```

Sous Eclipse

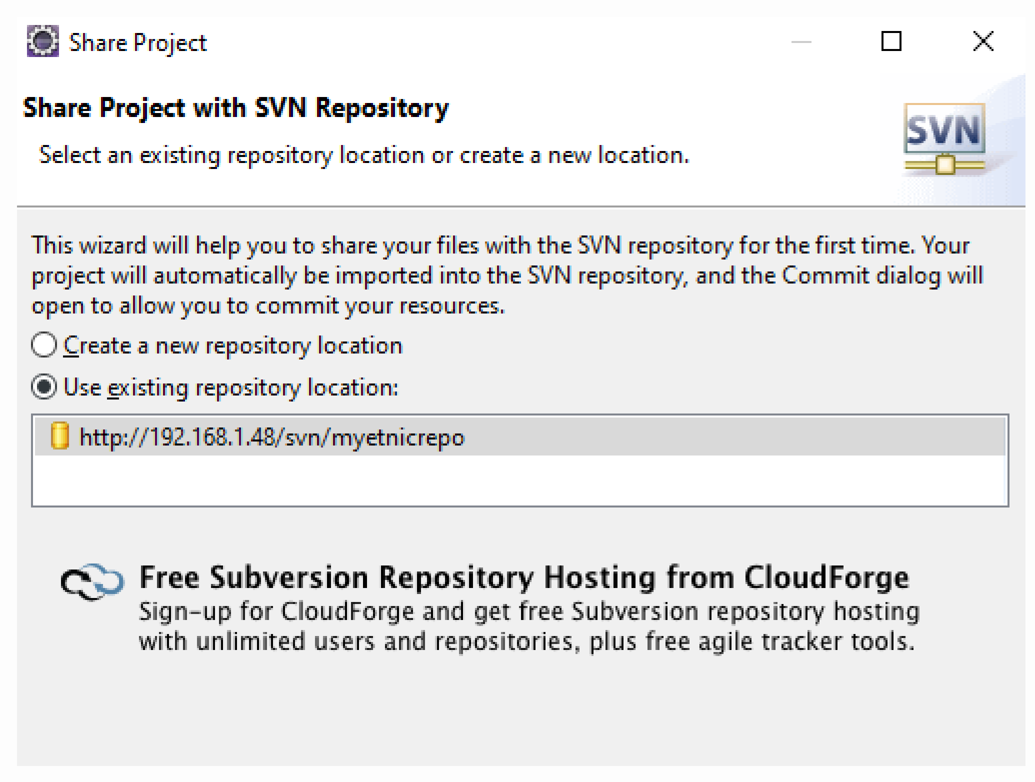
- La perspective SVN permet de « monter » le repository

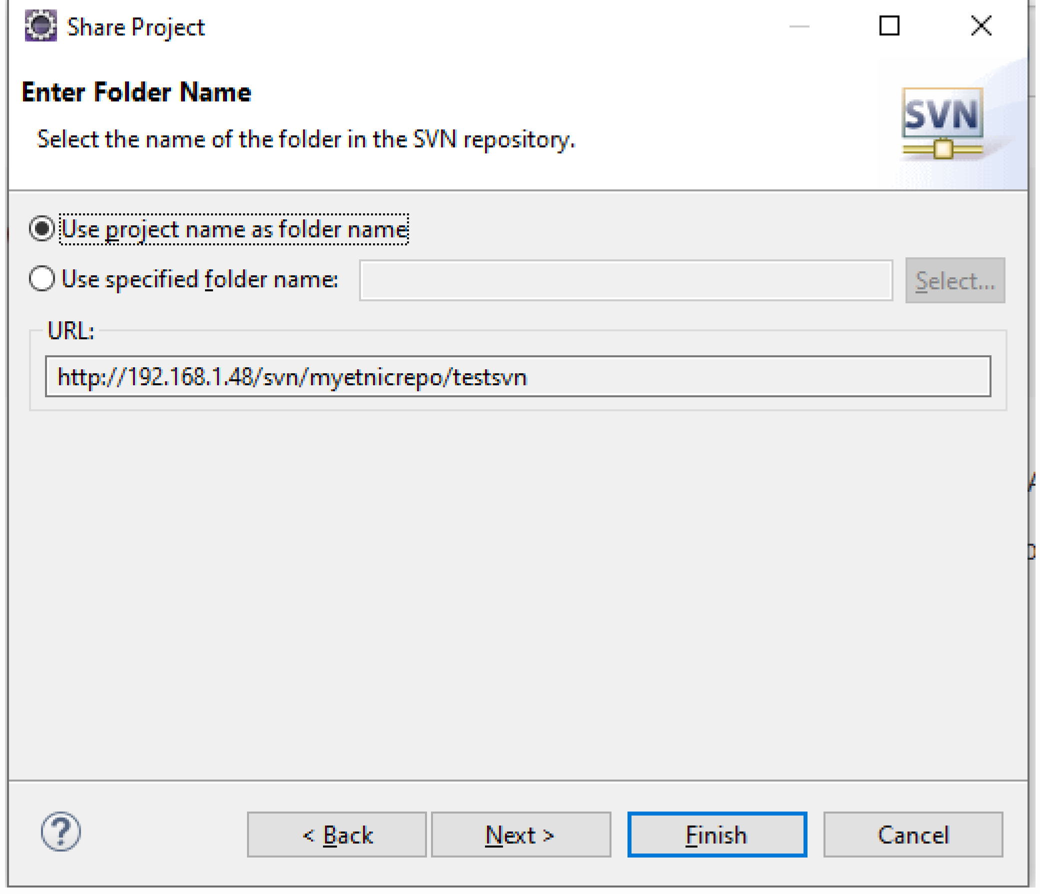
- Puis on monte le repository



Partage de projet

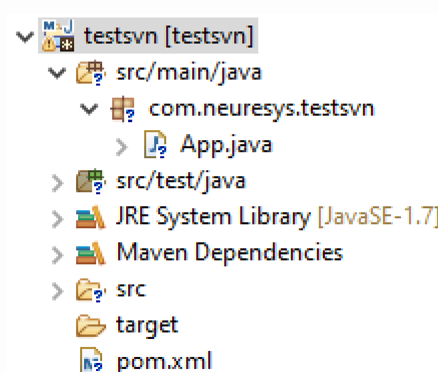
- L’option shared project permet de partager un projet vers SVN





Partage de projet

- La première chose que l'on remarque, c'est un petit point d'interrogation sur l'icône du fichier et la même sur celle du package, cela veut tout simplement dire que ces fichiers ne se trouvent pas sur le repository.



- Il faut pour cela faire le commit

```
pilou@qt:~$ svn list http://localhost/svn/myetnicrepo/ --username
'alice'
testsvn/
pilou@qt:~$ svn list http://localhost/svn/myetnicrepo/testsvn --
username 'alice'
```

Faire le commit

Changes — ☑ Show unversioned files ☐ Keep locks

- ☑ testsvn
  - ☑ .classpath
  - ☑ .project
  - ☑ pom.xml
- ☑ testsvn/.settings
  - ☑ org.eclipse.core.resources.prefs
  - ☑ org.eclipse.jdt.core.prefs
  - ☑ org.eclipse.m2e.core.prefs
- ☑ testsvn/src
- ☑ testsvn/src/main
- ☑ testsvn/src/main/java
- ☑ testsvn/src/main/java/com
- ☑ testsvn/src/main/java/com/neuresys
- ☑ testsvn/src/main/java/com/neuresys/testsvn
  - ☑ App.java
- ☑ testsvn/src/test
- ☑ testsvn/src/test/java
- ☑ testsvn/src/test/java/com
- ☑ testsvn/src/test/java/com/neuresys
- ☑ testsvn/src/test/java/com/neuresys/testsvn
  - ☑ AppTest.java

OK Cancel

Après le commit

Pour faire cela, rien de plus simple, il vous suffit de cliquer droit sur votre fichier, d'aller dans le menu team et de choisir "commit". Ensuite, il vous suffit d'entrer un commentaire de commit (c'est très pratique ensuite pour s'y retrouver avec des nombreuses versions) et de sélectionner le fichier dans la liste.

```
testsvn [testsvn]
  src/main/java
    com.neuresys.testsvn
      App.java 2  11/10/2019 10:15  svnadmin
  src/test/java
  JRE System Library [JavaSE-1.7]
  Maven Dependencies
  src
  target
  pom.xml 2  11/10/2019 10:15  svnadmin
```

```
pilou@qt:~$ svn list http://localhost/svn/myetnicrepo/testsvn --username 'alice'
.classpath
.project
.settings/
pom.xml
src/
```

Faire une modification et un commit

- La modification fait apparaitre une etoile dans eclise
- Apres un commit un 3 apparait en jaune (3 eme revision)

testsvn [testsvn]
src/main/java
com.neuresys.testsvn
App.java 2 11/10/2019 10:15 svnadmin
App 2 11/10/2019 10:15 svnadmin
main(String[]) : void
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml 2 11/10/2019 10:15 svnadmin

```
svn cat http://localhost/svn/myetnicrepo/testsvn/src/main/java/com/neuresys/testsvn/App.java --username 'alice'
package com.neuresys.testsvn;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
        System.out.println("From SVN");
    }
}
```

Récupération d’un projet

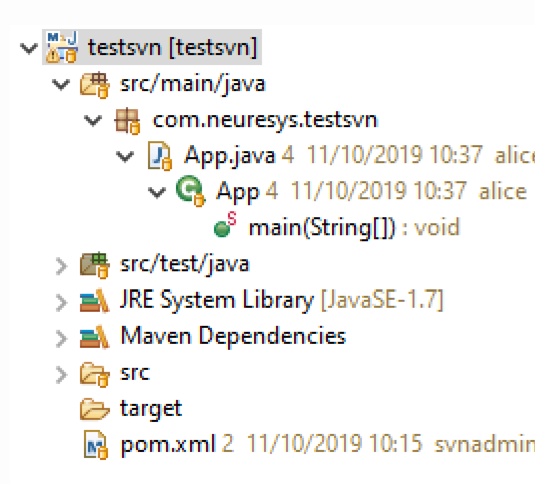
- La récupération d’un projet se fait via la commande checkout

```
svn checkout http://localhost/svn/myetnicrepo/testsvn --username 'alice'
A    testsvn/.settings
A    testsvn/src
A    testsvn/src/main
A    testsvn/src/main/java
A    testsvn/src/main/java/com
A    testsvn/src/main/java/com/neuresys
A    testsvn/src/main/java/com/neuresys/testsvn
A    testsvn/src/test
A    testsvn/src/test/java
A    testsvn/src/test/java/com
A    testsvn/src/test/java/com/neuresys
A    testsvn/src/test/java/com/neuresys/testsvn
A    testsvn/.classpath
A    testsvn/.project
A    testsvn/.settings/org.eclipse.core.resources.prefs
A    testsvn/.settings/org.eclipse.jdt.core.prefs
A    testsvn/.settings/org.eclipse.m2e.core.prefs
A    testsvn/pom.xml
A    testsvn/src/main/java/com/neuresys/testsvn/App.java
A    testsvn/src/test/java/com/neuresys/testsvn/AppTest.java
Checked out revision 3.
```

Update

```
svn commit  --username 'alice'

Log message unchanged or not specified
(a)bort, (c)ontinue, (e)dit:
c
Sending        src/main/java/com/neuresys/testsvn/App.java
Transmitting file data .done
Committing transaction...
Committed revision 4.
```

- Si alice fait un commit, il faut faire un update soit to head (dernière version) soit vers une version particulière



Provoquons un conflit

- Bob et Alice modifie le même fichier
- Alice fait un commit et Bob essaye de faire son commit
- Eclipse refuse le commit

```
svn: Commit failed (details follow):
svn: File
'D:\MyJavaProjects\classactions\testsvn\testsvn\src\main\java\com\neuresys\te
stsvn\App.java' is out of date
Item is out of date
svn: While preparing
'D:\MyJavaProjects\classactions\testsvn\testsvn\src\main\java\com\neuresys\te
stsvn\App.java' for commit
svn: File '/testsvn/src/main/java/com/neuresys/testsvn/App.java' is out of
date
```

Editons le conflit

Java Structure Compare

- Compilation Unit
  - App
    - main(String [])

Java Source Compare

Merged - App.java

```java
package com.neuresys.testsvn;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
        System.out.println("From SVN");
        System.out.println("Conflit par Bob");
    }
}
```

Theirs - App.java.r5

```java
package com.neuresys.testsvn;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
        System.out.println("From SVN");
        System.out.println("Conflit par Alice");

    }
}
```

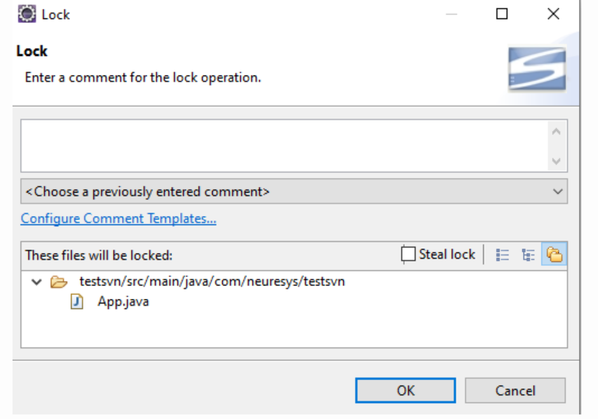
Résolution de conflit

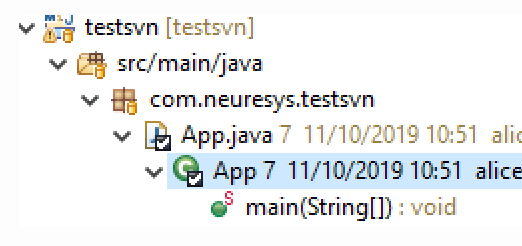
- Marquer le fichier comme étant résolue (clique droit dans le fichier) et commit

```
pilou@qt:~/myetnicproject/testsvn$ svn update  --username 'alice'
Updating '.':
U    src/main/java/com/neuresys/testsvn/App.java
Updated to revision 6.
```

Création d’un lock

- Nous allons mettre en place un lock afin d’éviter une modification






```
pilou@qt:~/myetnicproject/testsvn$ svn commit  --username 'alice'

Log message unchanged or not specified
(a)bort, (c)ontinue, (e)dit:
c
Sending        src/main/java/com/neuresys/testsvn/App.java
Transmitting file data .svn: E195022: Commit failed (details follow):
svn: E195022: File '/home/pilou/myetnicproject/testsvn/src/main/java/com/neuresys/testsvn/App.java' is
locked in another working copy
svn: E160038: While preparing
'/home/pilou/myetnicproject/testsvn/src/main/java/com/neuresys/testsvn/App.java' for commit
svn: E160038: '/svn/myetnicrepo/!svn/txr/7-8/testsvn/src/main/java/com/neuresys/testsvn/App.java': no
lock token available
```

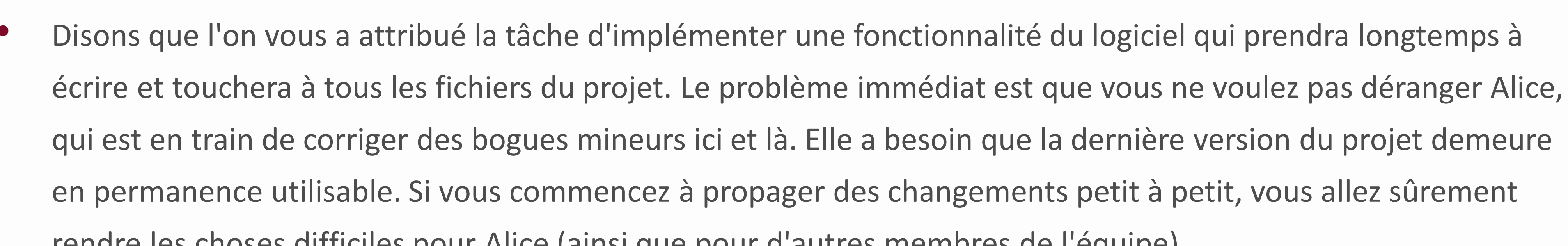
La gestion des branches

Démarche SVN

- La bonne démarche SVN est d’avoir le projet dans un répertoire /<nom du projet>/trunk. Il s’agit d’une convention expliquant qu’il s’agit du tronc de base.
- Il est important de faire des commits régulier afin 1 de pas perdre son travail, 2 éviter des fusions trop complexe.

Branche

- Comme avant, supposons que Alice et vous avez tous deux une copie de travail du projet « test_svn ». Plus spécifiquement, vous avez chacun une copie de travail de /test_svn/trunk. Tous les fichiers du projet sont dans ce sous-dossier plutôt que dans /test_svn lui-même, parce que votre équipe a décidé que la « ligne principale » de développement du projet allait se situer dans /test_svn/trunk.



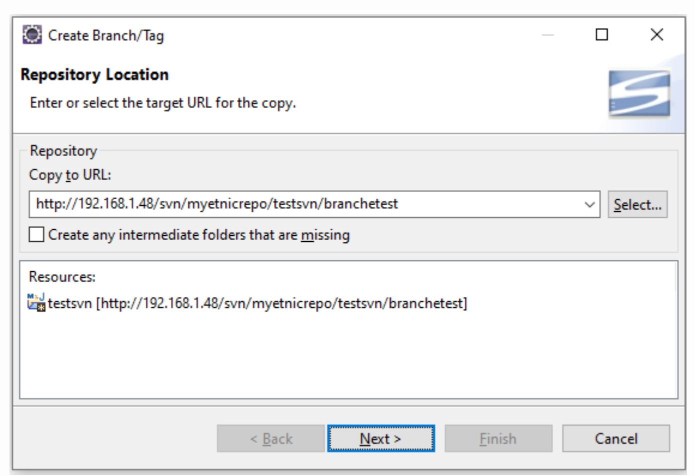
- Disons que l'on vous a attribué la tâche d'implémenter une fonctionnalité du logiciel qui prendra longtemps à écrire et touchera à tous les fichiers du projet. Le problème immédiat est que vous ne voulez pas déranger Alice, qui est en train de corriger des bogues mineurs ici et là. Elle a besoin que la dernière version du projet demeure en permanence utilisable. Si vous commencez à propager des changements petit à petit, vous allez sûrement rendre les choses difficiles pour Alice (ainsi que pour d'autres membres de l'équipe).

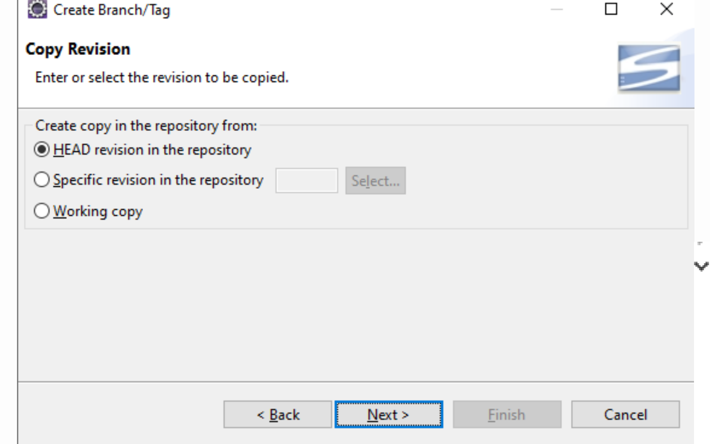
- Une stratégie possible est de vous isoler : vous pouvez arrêter de partager des informations avec Alice pendant une semaine ou deux. C'est-à-dire commencer à modifier et à réorganiser les fichiers dans votre copie de travail, mais sans effectuer de propagation ni de mise à jour avant que vous n'ayez complètement terminé la tâche.

Branche

- Une solution bien meilleure est de créer votre propre branche, ou ligne de développement, dans le dépôt. Ceci vous permettra de sauvegarder fréquemment votre travail un peu boiteux sans interférer avec vos collaborateurs ; vous pourrez toutefois partager une sélection d'informations avec eux. Vous découvrirez comment tout cela fonctionne exactement au fur et à mesure de ce chapitre.

- Créer une branche est très simple : il s'agit juste de faire une copie du projet dans le dépôt avec la commande **svn copy**.





testsvn [testsvn/branchetest]
src/main/java
com.neuresys.testsvn
App.java 7 11/10/2019 10:51 alice
App 7 11/10/2019 10:51 alice
main(String[]) : void
src/test/java
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml 2 11/10/2019 10:15 svnadmin
Untitled 1

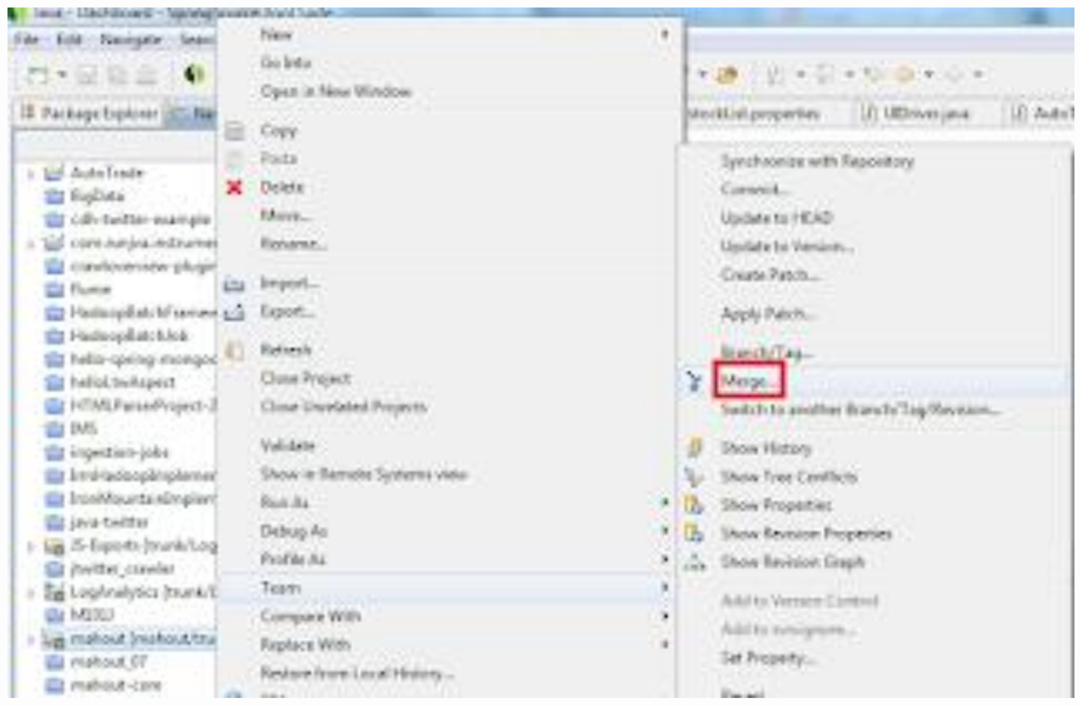
Fusion de branche

- La fusion de branche passe par la fonctionnalité merge branch d'eclipse.

1. Tout d'abord, assurez-vous d'être à jour. Mettez à jour votre copie de travail de la branche cible, c.-à-d. où vous êtes en train de fusionner. Dans cet exemple, nous travaillons sur le tronc de "core" et souhaitons saisir les modifications survenues dans la branche maintenance et les fusionner.
2. Résoudre les conflits éventuels. Il ne devrait y avoir aucun conflit à ce stade entre la copie de travail et le référentiel.
3. Sélectionnez l'option Fusion SVN sur la copie de travail. Dans Eclipse, cela se trouve dans le menu "Equipe" et s'appelle "Fusionner une branche". SVN: Fusion dans Eclipse
4. Changez l'URL de l'expéditeur vers la branche spécifique que vous souhaitez fusionner dans votre copie de travail.
5. Changez la révision de la dernière révision fusionnée dans la branche cible. Essentiellement, vous ne souhaitez pas fusionner tout l'historique de la branche, mais simplement inclure ces modifications depuis la dernière fusion. Il n'existe pas de moyen facile de déterminer le dernier point de fusion dans Subversion. Vous devez consulter votre journal des messages et rechercher le dernier commit qui parle de fusion. Si vous êtes discipliné à propos des messages de commit que vous utilisez pour la fusion, cela devrait être facile (voir ci-dessous). Notez la nature de cette révision. Vous en aurez besoin ultérieurement lorsque vous validez vos modifications. SVN: fusionner avec Eclipse
6. Changez la dernière révision à la dernière (c'est-à-dire la tête). Notez la nature de cette révision. Vous en aurez besoin ultérieurement lorsque vous validez vos modifications.
7. Cliquez sur Fusionner et attendez. Selon la taille des différences, cela peut être rapide ou Eclipse peut tomber. Si vous avez un changement tellement énorme que vous ne pouvez pas le faire dans Eclipse, vous devrez peut-être réduire la plage de révisions que vous fusionnez. Ou vous devrez peut-être ignorer certaines révisions et les faire manuellement si elles sont volumineuses. Nous avons eu ce problème de temps en temps lors de la mise à jour de grandes bibliothèques tierces. La grande majorité du temps, tout ira bien.
8. Vérifiez les modifications et résolvez les conflits. Une fois la fusion terminée, examinez les modifications apportées à votre copie de travail et assurez-vous de résoudre tout conflit éventuel.
9. Une fois que tous les changements ont été résolus dans la copie de travail cible, archivez-les avec un seul commit. La raison pour laquelle vous ne faites pas beaucoup de commits est que ce sont des modifications qui auraient dû être documentées dans la branche à partir de laquelle vous avez fusionné. Le message de validation doit être dans un format spécifique qui détaille la fusion et qui est facile à trouver à l'avenir. Nous utilisons le format suivant, mais vous pouvez utiliser tout ce qui vous convient, à condition de vous y tenir.

Fusion de Branches

- Dans eclipse Cliquez avec le bouton droit de la souris sur le projet ou le dossier sur lequel vous souhaitez fusionner. Dans le menu contextuel, sélectionnez Equipe-> Fusionner comme indiqué dans la capture d'écran ci-dessous.



Fusion de Branches

- Cela fait apparaître ci-dessous l’assistant de fusion, à partir duquel vous pouvez choisir le type de fusion que vous souhaitez effectuer.
- Merge a rank of revision: Utilisez cette méthode pour effectuer des fusions en aval sur une branche à partir d’une autre branche ou d’une autre ligne. Généralement, les modifications d'une branche plus ancienne sont fusionnées vers une nouvelle branche.
- Dans ce cas, vous devez spécifier la plage de révisions à fusionner. Si par le passé vous avez fusionné jusqu'à la version 102, dans la commande de fusion, votre point de révision de départ doit être 103, comme indiqué ci-dessous.

Merge a rank of revision

```
# svn merge http://192.168.101.1/Repository/branch/KSS/BUILD/SERVER/conf/kss
-r103:HEAD
U    src/Client.java
U    src/Stub.java
U    src/Ims.java
A    src/Soap.java
A    src/SoapInterface.java
U    src/imsInterfaec.java
U    src/SoapInterface.Java
```

- Le journal de sortie affiche les fichiers avec leur statut:

- "U" signifie que ce fichier a été mis à jour, c'est-à-dire que les modifications de la copie de travail sont conservées et que les modifications de B ont été ajoutées à la copie de travail locale.

- ‘A’ signifie que tout fichier ajouté au dossier de la copie de travail du B a été ajouté à votre dossier de la copie de travail.

Merge a rank of revision

```
# svn merge http://192.168.101.1/Repository/branch/KSS/BUILD/SERVER/conf/kss
-r103:HEAD
U    src/Client.java
U    src/Stub.java
C    src/Ims.java
C    src/Soap.java
A    src/SoapInterface.java
U    src/imsInterfaec.java
U    src/SoapInterface.Java
```

- Dans le journal de sortie ci-dessus, l'état «C» avant le fichier Ims.java, Soap.java signifie CONFLICT.
- C'est un cas de conflit de fichier, cela signifie que dans ce fichier, l'utilisateur A et l'utilisateur B ont modifié les mêmes lignes de code. Dans ce cas, SVN ne sait pas quelle version du fichier il doit conserver.
- Vous devez résoudre ce problème de conflit. c'est-à-dire, ouvrez manuellement les deux fichiers et voyez quelle version vous souhaitez conserver, et ce ne sera qu'ensuite que la fusion sera terminée.

Merge of two Branch

- Supposons qu'il existe un utilisateur «C» qui souhaite compiler le projet complet. L'utilisateur C doit obtenir les modifications effectuées par «A» et «B» dans sa copie de travail locale, en exécutant la commande suivante à partir du dossier de la copie de travail de C

```
# svn merge
http://192.168.101.1/Repository/branch/user/HSS/BUILD/SERVER/uma/
http://192.168.101.1/Repository/branch/user2/HSS/BUILDSERVER/uma/
```

Introduction à la programmation avec Java

- Les fondements de la programmation
- Les constructions de base d'un programmes
- Tests Logiciels
- Bonne Pratiques de conception d'une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d'un projet avec Maven
  - Définition de la structure d'un projet.
  - Les conventions. Les dépendances entre projets. Les tâches prédéfinies : compilation, génération d'archives...
  - Les perspectives Maven proposées par les plug-ins Eclipse.
- Quelques Design Pattern
- Mesure de la qualité

Définition de la structure d’un projet avec Maven

Définition de la structure d’un projet

- Qu’est ce que Maven?
- Convention Maven.

Apache Maven

- La création d'un projet logiciel comprend généralement des tâches telles que le téléchargement de dépendances, l'ajout de fichiers JAR sur un chemin d'accès aux classes, la compilation du code source en code binaire, l'exécution de tests, l'intégration du code compilé dans des artefacts déployables tels que les fichiers JAR, WAR et ZIP, et le déploiement de ces artefacts. sur un serveur d'applications ou un référentiel.

- Apache Maven automatise ces tâches en minimisant les risques d'erreur humaine lors de la construction manuelle du logiciel et en séparant le travail de compilation et d'empaquetage de notre code de celui de la construction de code.

- Maven utilise un système d’un information central - le modèle POM (Project Object Model) - écrite en XML.

Apache Maven

Les principales caractéristiques de Maven sont les suivantes:

- configuration de projet simple qui suit les meilleures pratiques: Maven tente d'éviter autant que possible la configuration en fournissant des modèles de projet (nommés archétypes)
- gestion des dépendances: elle inclut la mise à jour automatique, le téléchargement et la validation de la compatibilité, ainsi que le signalement des fermetures de dépendance (également appelées dépendances transitives)
- isolement entre les dépendances du projet et les plugins: avec Maven, les dépendances du projet sont extraites des référentiels de dépendances, tandis que les dépendances des plugins sont extraites des référentiels de plugins, ce qui réduit le nombre de conflits lorsque les plugins commencent à télécharger des dépendances supplémentaires.
- système de référentiel central: les dépendances de projet peuvent être chargées à partir du système de fichiers local ou de référentiels publics, tels que Maven Central

Apache Maven

Maven repose sur l'utilisation de plusieurs concepts :

- Les artéfacts : composants identifiés de manière unique
- **Le principe de convention over configuration : utilisation de conventions par défaut pour standardiser les projets**
- Le cycle de vie et les phases : les étapes de construction d'un projet sont standardisées
- Les dépôts (local et distant)

POM

- La configuration d'un projet Maven s'effectue via un POM (Project Object Model), représenté par un fichier pom.xml. Le POM décrit le projet, gère les dépendances et configure les plugins pour la construction du logiciel.

- Le POM définit également les relations entre les modules de projets multi-modules.

POM.xml

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>be.etnic</groupId>
    <artifactId>monjob</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>be.etnic.monjob</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
            //...
            </plugin>
        </plugins>
    </build>
</project>
```

Convention plutôt que configuration

Maven met en oeuvre le principe de convention over configuration pour utiliser par défaut les mêmes conventions.

Répertoire	Contenu
/src	les sources du projet (répertoire qui doit être ajouté dans le gestionnaire de sources)
/src/main	les fichiers sources principaux
/src/main/java	le code source (sera compilé dans /target/classses)
/src/main/resources	les fichiers de ressources (fichiers de configuration, images, ...). Le contenu de ce répertoire est copié dans target/classes pour être inclus dans l'artéfact généré
/src/main/webapp	les fichiers de la webapp
/src/test	les fichiers pour les tests
/src/test/java	le code source des tests (sera compilé dans /target/test-classses)
/src/test/resources	les fichiers de ressources pour les tests
/target	les fichiers générés pour les artéfacts et les tests (ce répertoire ne doit pas être inclus dans le gestionnaire de sources)
/target/classes	les classes compilées
/target/test-classes	les classes compilées des tests unitaires
/target/site	site web contenant les rapports générés et des informations sur le projet
/pom.xml	le fichier POM de description du projet

Partie Projet

Maven utilise un ensemble d'identifiants, également appelés coordonnées, pour identifier de manière unique un projet et spécifier la manière dont l'artefact de projet doit être empaqueté:

- groupId - un nom de base unique de la société ou du groupe qui a créé le projet
- artifactId - un nom unique du projet
- version - une version du projet
- packaging - une méthode d’emballage (par exemple, WAR / JAR / ZIP)

Les trois premiers (groupId: artifactId: version) se combinent pour former l'identificateur unique et constituent le mécanisme par lequel vous spécifiez les versions des bibliothèques externes (par exemple, les fichiers JAR) que votre projet utilisera.

Dépendances

Généralement un artéfact a besoin d'autres artéfacts qui sont alors désignés comme des dépendances présentant elles-mêmes des dépendances.

- Une dépendance peut être optionnelle : elle n'est requise que si une fonctionnalité particulière est utilisée. C'est par exemple le cas pour Hibernate avec le pool de connections c3p0 ou l'implémentation du cache de second niveau.
- Certaines dépendances ne sont utiles que pour certaines phases, par exemple lors de la phase de tests qui devrait être la seule à avoir besoin d'un framework pour les tests unitaires ou d'un framework pour le mocking.
- Certains artéfacts possèdent un support de plusieurs implémentations : c'est par exemple le cas de commons-logging qui a besoin d'une implémentation d'un moteur de logging. Il va dynamiquement utiliser celui qui sera trouvé.

La gestion des dépendances de Maven repose sur plusieurs concepts :

- les dépôts : permet de stocker les artéfacts
- la portée : permet de préciser dans quel contexte une dépendance est utilisée
- la transitivité : permet de gérer les dépendances de dépendances
- l'héritage

Dépendances

Les bibliothèques externes utilisées par un projet s'appellent donc des dépendances. La fonctionnalité de gestion des dépendances de Maven assure le téléchargement automatique de ces bibliothèques à partir d'un référentiel central, vous évitant ainsi de les stocker localement.

- Ceci est une fonctionnalité clé de Maven et offre les avantages suivants:
- utilise moins d'espace de stockage en réduisant considérablement le nombre de téléchargements hors des référentiels distants
- rend la vérification d'un projet plus rapide
- fournit une plate-forme efficace pour l'échange d'artefacts binaires au sein de votre organisation et au-delà, sans qu'il soit nécessaire de créer un artefact à partir de la source à chaque fois

Afin de déclarer une dépendance à une bibliothèque externe, vous devez fournir le groupId, artifactId et la version de la bibliothèque. Jetons un coup d'oeil à un exemple:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.3.5.RELEASE</version>
</dependency>
```

Les dépots

Maven utilise la notion de référentiel ou dépôt (repository) pour stocker les dépendances et les plugins requis pour générer les projets.

Un dépôt contient un ensemble d'artéfacts qui peuvent être des livrables, des dépendances, des plugins, ... Ceci permet de centraliser ces éléments qui sont généralement utilisés dans plusieurs projets : c'est notamment le cas pour les plugins et les dépendances.

Maven distingue deux types de dépôts : local et distant (remote). Ces dépôts peuvent être gérés à plusieurs niveaux :

- dépôt central : il stocke des dépendances et les plugins utilisables par tout le monde car disponible sur le web; ce sont généralement des artéfacts open source
- dépôt local : il stocke une copie des dépendances et plugins requis par les projets à générer en local. Ces artéfacts sont soit téléchargés des dépôts centraux soit créés avec Maven
- un référentiel au niveau entreprise ou domaine : ce référentiel permet de limiter les dépendances et les plugins ainsi que leurs versions à celles qui sont utilisables en local. Il permet de gérer et de restreindre les dépendances pour un certain niveau (entreprise, domaine, service, équipe, projet ...). Son utilisation est requise lorsque le nombre de projets et leur complexité sont importants.
- Maven utilise une structure de répertoires particulière pour organiser le contenu d'un référentiel et lui permettre de retrouver les éléments requis :
- Chemin_referentiel/groupId/artifactId/version

Les dépots

Par exemple avec Junit 3.8.2, dont les identifiants sont :

- <groupId>junit</groupId>
- <artifactId>junit</artifactId>
- <version>3.8.2</version>

- La structure de répertoires dans le dépôt est :
- \junit\junit\3.8.2

Le répertoire de la version contient au moins l'artefact et son POM mais il peut aussi éventuellement contenir d'autres fichiers liés contenant une archive avec les sources, la Javadoc, la valeur du message digest calculée avec SHA-1, ...

Les dépots

- Un référentiel dans Maven est utilisé pour contenir des artefacts de construction et des dépendances de types variés. Le référentiel local par défaut se trouve dans le dossier .m2 / repository, sous le répertoire de base de l'utilisateur.
- Si un artefact ou un plug-in est disponible dans le référentiel local, Maven l'utilise. Sinon, il est téléchargé depuis un référentiel central et stocké dans le référentiel local. Le référentiel central par défaut est Maven Central.
- Certaines bibliothèques, telles que le serveur JBoss, ne sont pas disponibles dans le référentiel central mais sont disponibles dans un autre référentiel. Pour ces bibliothèques, vous devez fournir l'URL du référentiel alternatif dans le fichier pom.xml:

```
<repositories>
    <repository>
        <id>JBoss repository</id>
<url>http://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
</repositories>
```

Les propriétés

- Les propriétés personnalisées peuvent aider à rendre votre fichier pom.xml plus facile à lire et à gérer. Dans le cas d'utilisation classique, vous utiliseriez des propriétés personnalisées pour définir des versions pour les dépendances de votre projet.

- Les propriétés Maven sont des espaces réservés aux valeurs et sont accessibles n'importe où dans un fichier pom.xml en utilisant la notation $ {name}, où nom est la propriété.

```
<properties>
    <spring.version>4.3.5.RELEASE</spring.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
```

Définition de la structure d'un projet avec Maven

Cycle de construction

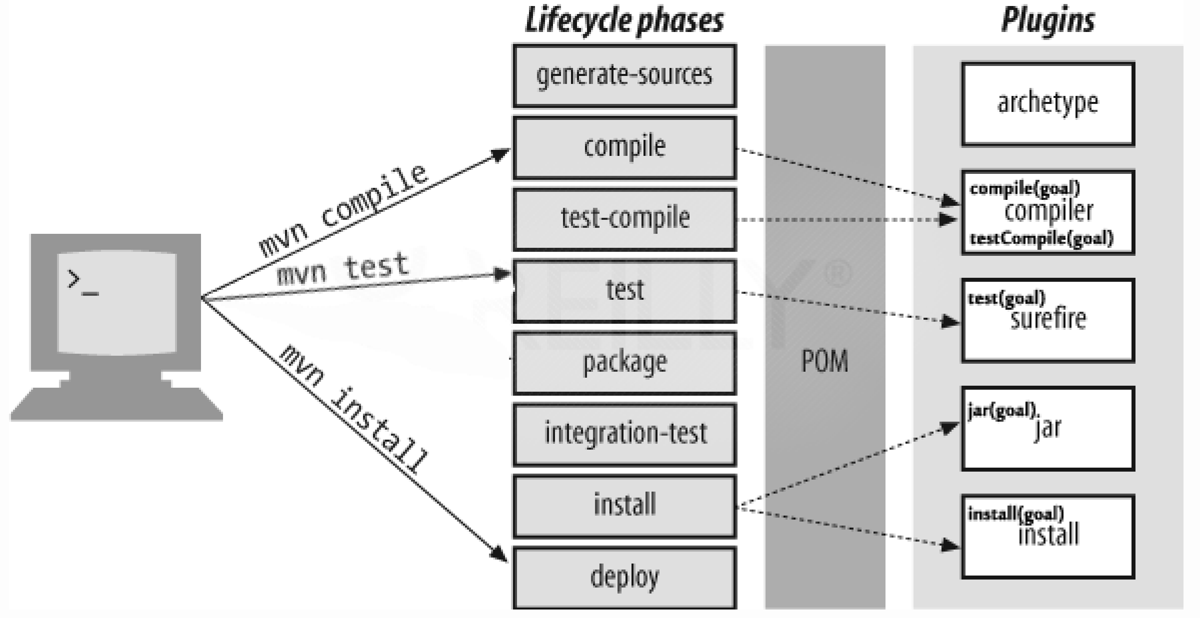
- Qu’est ce qu’un cycle de construction?
- Quel sont les phases/goals classiques?

Maven définit 4 éléments d'un processus de construction:

- Cycle de la vie Trois cycles de vie intégrés (ou cycles de vie de génération): defaults, clean, site.
- Phase Chaque cycle de vie est composé de phases, par ex. pour le cycle de vie par défaut: compile, test, update, install, etc.
- Plugin Un artefact qui fournit un ou plusieurs objectifs. En fonction du type de packaging (jar, war, etc.), les objectifs des plugins sont liés à des phases par défaut. (Liaisons de cycle de vie intégrées)
- goal La tâche (action) qui est exécutée. Un plugin peut avoir un ou plusieurs goals.
- Un ou plusieurs goals doivent être spécifiés lors de la configuration d'un plug-in dans un POM. De plus, si un plugin n'a pas défini de phase par défaut, le ou les objectifs spécifiés peuvent être liés à une phase.

Schéma Maven

- Lors du choix d'une phase, les phases sont enchainé. Chaque phase utilise ensuite des plugins pour exécuter leurs taches.



Build Life Cycles

- Maven est basé sur le concept central de Build Life Cycles. Dans chaque cycle de vie de construction, il y a des phases de construction et dans chaque phase de construction, des objectifs de construction.

- Nous pouvons exécuter une phase de construction ou un objectif de construction. Lors de l'exécution d'une phase de construction, nous exécutons tous les objectifs de construction de cette phase. Les objectifs de construction sont affectés à une ou plusieurs phases de construction. Nous pouvons également exécuter directement un objectif de construction.
- Il existe trois principaux cycles de vie de construction intégrés:
  - default
  - clean
  - site

Build Phase

- Chaque construction Maven suit un cycle de vie spécifique. Vous pouvez exécuter plusieurs objectifs de cycle de vie de la génération, notamment ceux de compiler le code du projet, de créer un package et d’installer le fichier archive dans le référentiel de dépendances Maven local.

- La liste suivante répertorie les principales build phase Maven pour le build cycle default:
  - validate - vérifie l'exactitude du projet
  - compile - compile le code source fourni en artefacts binaires
  - test - exécute les tests unitaires
  - package - compile le code compilé dans un fichier archive
  - integration-test - exécute des tests supplémentaires, qui nécessitent le package
  - verify - vérifie si le paquet est valide
  - install - installe le fichier de package dans le référentiel Maven local
  - deploy - déploie le fichier de package sur un serveur ou un référentiel distant

Build Phase

Donc, pour passer par les phases ci-dessus, il suffit d'appeler une commande:

- mvn <phase> {Ex: mvn install}

Pour la commande ci-dessus, à partir de la première phase, toutes les phases sont exécutées de manière séquentielle jusqu'à la phase d'installation.

Plugin

- Maven en lui-même n'est composé que d'un noyau très léger.
- Toutes les fonctionnalités pour générer un projet sont sous la forme de plugins qui doivent être présents dans le référentiel local ou téléchargés lors de la première utilisation.
- La déclaration et la configuration des plugins à utiliser se fait dans le fichier POM.
- Certains plugins utilisés dans le cycle de vie par défaut n'ont pas besoin d'être définit explicitement dans le fichier POM, sauf si la configuration par défaut doit être modifiée.
- Il est aussi possible de développer ses propres plugins.
- Un plugin est un artéfact Maven : il est donc identifié par un groupId, un artifactId et un numéro de version (par défaut, la version la plus récente).
- La personnalisation d'un projet se fait en utilisant et en configurant des plugins.

Plugin

- Les plugins sont des dépendances qui sont stockées dans le dépôt local après avoir été téléchargées.

- Le tag <configuration> permet de fournir des paramètres qui seront utilisés par le plugin.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.0.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Définition de la structure d'un projet avec Maven

Exemple diverse

- L’idée est de voir des exemples de pom
- Enfin l’intégration avec Eclipse seras montrée

maven-archetype-quickstart

- Nous pouvons créer un exemple simple maven en exécutant l'archétype: générer la commande de l'outil mvn.

- Pour créer un projet Java simple à l'aide de maven, vous pouvez ouvrir une invite de commande et exécuter l'archétype: générate de mvn.

maven-archetype-quickstart

- mvn archetype:generate -DgroupId=groupid -DartifactId=artifactid -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=booleanValue

```
Mvn archetype:generate -DgroupId=be.etnic -DartifactId=myjob   -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: D:\MyJavaProjects\mvnTutorial
[INFO] Parameter: package, Value: be.etnic
[INFO] Parameter: groupId, Value: be.etnic
[INFO] Parameter: artifactId, Value: myjob
[INFO] Parameter: packageName, Value: be.etnic
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: D:\MyJavaProjects\mvnTutorial\myjob
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.077 s
[INFO] Finished at: 2019-10-25T11:23:14+02:00
[INFO] ------------------------------------------------------------------------
```

maven-archetype-quickstart

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion> <groupId>be.etnic</groupId>  <artifactId>myjob</artifactId>  <packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>  <name>myjob</name>

  <url>http://maven.apache.org</url>

  <dependencies>

    <dependency>     <groupId>junit</groupId>     <artifactId>junit</artifactId>      <version>3.8.1</version>     <scope>test</scope>

    </dependency>   </dependencies></project>
```

```
myjob
    │   pom.xml
    │
    └───src
        ├───main
        │   └───java
        │       └───be
        │           └───etnic
        │                   App.java
        │
        └───test
            └───java
                └───be
                    └───etnic
                            AppTest.java
```

maven-archetype-quickstart

- Nous allons rajouter quelques propriétés
- Le plugin exec qui permet via un mvn exec:exec d’executer un executable Java

maven-archetype-quickstart

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>be.etnic</groupId>  <artifactId>myjob</artifactId>  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>myjob</name>
    <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
   <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.6.0</version>
                <configuration>
                      <executable>java</executable>
          <arguments>
            <argument>-classpath</argument>
            <classpath/>
            <argument>be.etnic.App</argument>
             </arguments>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

wildfly-javaee7-webapp-archetype

- Redhat fournit un plugin pour générer des projets wildfly/Jboss facilement.
- Nous allons utiliser ici aussi l'option génératе avec le dit plugin.

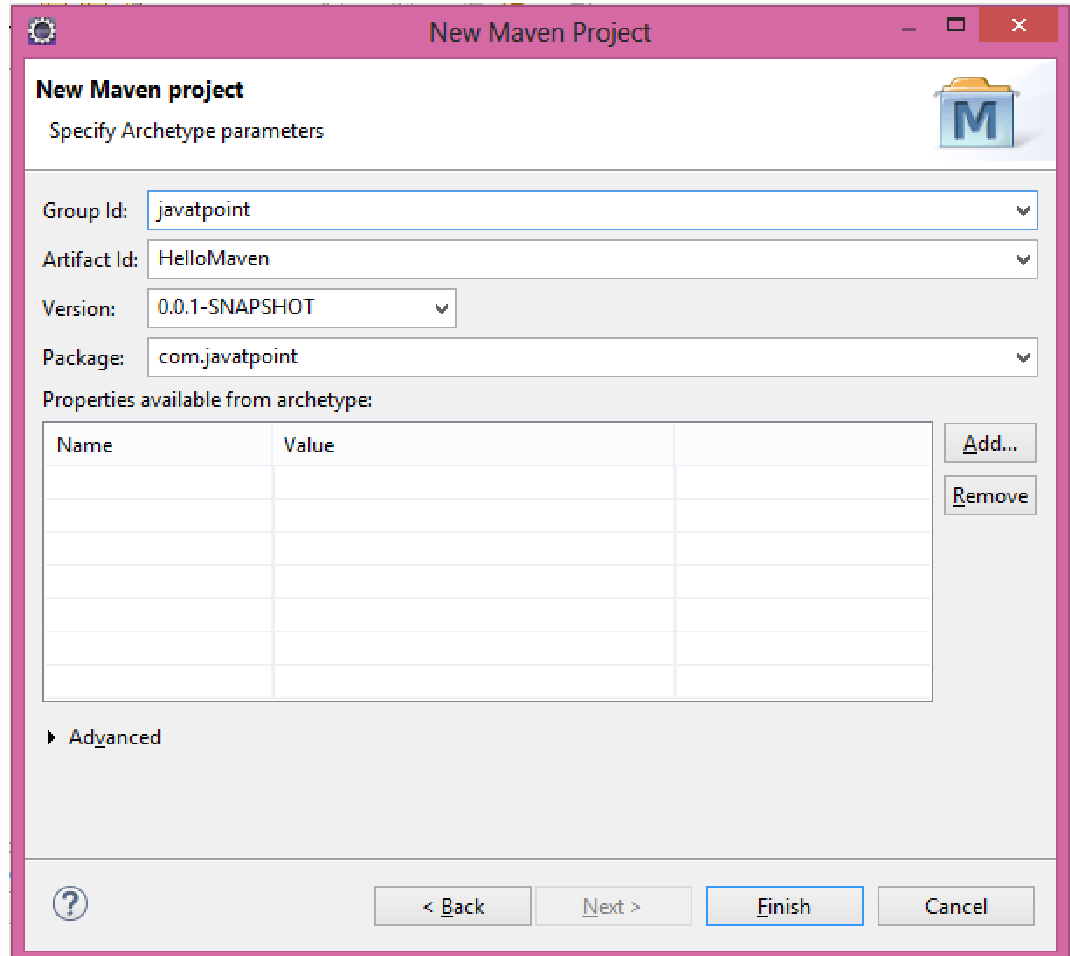
```
mvn -DarchetypeGroupId=org.wildfly.archetype -
DarchetypeArtifactId=wildfly-javaee7-webapp-archetype -
DarchetypeVersion=8.2.0.Final -
DarchetypeRepository=https://nexus.codehaus.org/content/repositor
ies/snapshots/ -DgroupId=com.sample -DartifactId=wildfly-sample -
Dversion=1.0-SNAPSHOT -Dpackage=com.sample -
Darchetype.interactive=false --batch-mode --update-snapshots
archetype:generate
```

wildfly-javaee7-webapp-archetype

```
pom.xml
│   README.md
└───src
    ├───main
    │   ├───java
    │   │   └───com
    │   │       └───sample
    │   │           ├───controller  MemberController.java
    │   │           ├───data        MemberListProducer.java   MemberRepository.java
    │   │           ├───model       Member.java
    │   │           ├───rest        JaxRsActivator.java       MemberResourceRESTService.java
    │   │           ├───service     MemberRegistration.java
    │   │           └───util        Resources.java
    │   ├───resources               import.sql
    │   │   │
    │   │   └───META-INF            persistence.xml
    │   └───webapp                  index.html                index.xhtml
    │       │
    │       ├───resources
    │       │   ├───css             screen.css
    │       │   └───gfx             asidebkg.png bkg-blkheader.png  dualbrand_logo.png headerbkg.png wildfly_400x130.jpg
    │       └───WEB-INF             beans.xml wildfly-sample-ds.xml
    │           └───templates       default.xhtml
    └───test
        ├───java
        │   └───com
        │       └───sample
        │           └───test        MemberRegistrationTest.java
        └───resources               arquillian.xml test-ds.xml
            └───META-INF            test-persistence.xml
```

Maven Eclipse

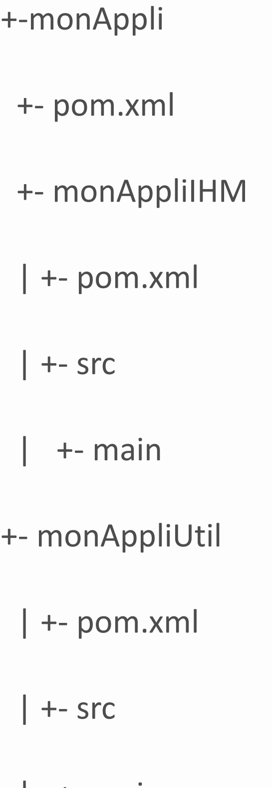
- Dans eclipse, Cliquer sur File menu → New → Project → Maven → Maven Project. → Next → Next → Next.
- Donner le groupId, ArtifactId et cliquer sur Finish



Multi Module

- Maven propose la possibilité d'utiliser des projets multi-modules.
- Un projet multi-modules est un projet qui contient d'autres projets fils : c'est un regroupement logique de sous-projets.
- Il contient un fichier pom.xml mais le projet ne crée pas lui-même d'artéfact.
- Il est possible d'imbriquer des projets multi-modules.

Exemple du projet MonAppli ayant deux sous projets IHM et Utils



```
+-monAppli
  +- pom.xml
  +- monAppliIHM
  | +- pom.xml
  | +- src
  |   +- main
+- monAppliUtil
  | +- pom.xml
  | +- src
  |   +- main
```

MonAppli

- Le fichier POM du projet multi-module doit avoir le type de packaging pom

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>be.etnic</groupId>
  <artifactId>monAppli</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>monAppli</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <modules>
    <module>monAppliIHM</module>
    <module>monAppliUtil</module>
  </modules>
</project>
```

MonAppli

- Le chemin de chacun des modules est précisé dans un tag <module> fils du tag <modules>. Cela indique à Maven que les opérations ne vont pas être réalisées sur le projet mais sur les modules du projet.
- Le POM de chacun des modules doit contenir un tag <parent> qui permet d'identifier le module parent.
- Le POM du module monAppliIHM

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>be.etnic</groupId>
    <artifactId>monAppli</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <groupId>be.etnic.monappli.util</groupId>
  <artifactId>monAppliUtil</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>monAppliUtil</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
```

Introduction à la programmation avec Java

- Les fondements de la programmation
- Les constructions de base d’un programmes
- Tests Logiciels
- Bonne Pratiques de conception d’une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d’un projet avec Maven
- Quelques Design Pattern
  - Les objectifs et les avantages.
  - Les Design Patterns les plus populaires dans les architectures logicielles modernes.
- Mesure de la qualité

Design Pattern

Les objectifs et les avantages

- Qu’est ce qu’un design pattern?
- Quels sont les avantages d’un pattern

- Formalisés dans le livre du « Gang of Four » (GoF, Erich Gamma, Richard Helm, Ralph Johnson (en) et John Vlissides (en)) intitulé Design Patterns – Elements of Reusable Object-Oriented Software9 en 1995.
- C’est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels1.

Pourquoi utiliser un design pattern?

- Pour accélérer le processus de développement en fournissant des paradigmes de développement éprouvés.
- Pour anticiper des problématiques qui peuvent ne devenir visibles que plus tard dans la mise en œuvre.
- Pour améliorer la lisibilité du code en fournissant une standardisation.

Pourquoi comprendre les design pattern en Java?

- Java/J2EE est une collection de pattern.
- En particulier J2EE est basé sur le pattern IoC
- Les singletons/Dao (accès aux bases) sont communément utilisé en Java
- L'idée est d'avoir les quelques patterns pour comprendre le monde J2EE.

Quelques Pattern

Delegation Pattern

- Qu’est ce que le Delegation Pattern (à quoi cela sert)
- Implémentation

Delegation Pattern

- Utiliser lorsqu’un objet de haut niveau (une imprimante par exemple) doit utiliser des classes techniques pour faire une fonctionnalité technique (ecrire en noir et blanc, ecrire en couleur …).
- Utilisé dans la notion de composition (une classe qui en utilise une autre) dans par exemple les classe IO de java

Delegation Pattern

- Le délégation pattern signifie transférer la responsabilité d'une tâche particulière à une autre classe ou méthode.
- C'est une technique dans laquelle un objet exprime certains comportements à l'extérieur mais délègue en réalité la responsabilité de la mise en œuvre de ce comportement à un objet associé.
- Utilisez la délégation pour atteindre les objectifs suivants
- Réduire le couplage des méthodes à leur classe
  - Les composants qui se comportent de manière identique, mais se rendent compte que cette situation peut changer dans le futur.
  - Si vous devez utiliser une fonctionnalité dans une autre classe mais que vous ne souhaitez pas la modifier, utilisez la délégation au lieu de l'héritage.

Delegation Pattern

```
class TicketBookingByAgent implements TravelBooking {

    TravelBooking t;

    public TicketBookingByAgent(TravelBooking t) {
        this.t = t;
    }

    // Delegation --- Here ticket booking responsibility
    // is delegated to other class using polymorphism
    @Override
    public void bookTicket() {
        t.bookTicket();
    }
}
```

```
interface TravelBooking {
    public void bookTicket();
}
```

```
class TrainBooking implements TravelBooking {
    @Override
    public void bookTicket() {
        System.out.println("Train ticket 
booked");
    }
}
```

```
class AirBooking implements TravelBooking {
    @Override
    public void bookTicket() {
        System.out.println("Flight ticket 
booked");
    }
}
```

```
public class DelegationDemonstration {

   public static void main(String[] args) {
    // Here TicketBookingByAgent class is internally
    // delegating train ticket booking responsibility to other 
class
        TicketBookingByAgent agent = new 
TicketBookingByAgent(new TrainBooking());
        agent.bookTicket();

        agent = new TicketBookingByAgent(new AirBooking());
        agent.bookTicket();
   }
}
```

Quelques Design Pattern

Factory Pattern

- Un des patterns les plus importants en JEE
- Centralisation des opérations de créations.

Factory Pattern

- C’est le pattern le plus usité en Java (c’est la base de l’IOC, de Spring ..)
- Plusieurs cas d’usage:
  - La création d’un objet est complexe (constructeur compliqué) et on souhaite caché cette complexité.
  - Le type exact de l’objet à créer n’est connue qu’a l’execution du programme
  - On suppose que le type exact de l’objet va changer et on souhaite centraliser les opérations de construction.

Centralisation des opérations de créations

- Soit l’interaction avec une Banque (Mercanet Atos).
- Si on fournit une classe Mercanet, elle seras construite partout et changer Mercanet par Hipay/Payline seras donc complexe
- La solution est la mise en place d’une factory

Centralisation des opérations de créations

```
class PaiementFactory
{
// initialise le paiement
Paiement createPaiment()
{
return new Mercanet(…);
}
}
```

```
Interface Paiement
{
// initialise le paiement
void initierPaiement(int amount);
}
```

```
public static void main(String [] args)
{
Paiement p=(new
PaiementFactory()).createPaiement()
}
```

```
Class Mercanet implements Paiement
{
Public Mercanet(String secretKey, …)
// initialise le paiement
void initierPaiement(int amount);
}
```

Factory Pattern

```java
public class ShapeFactory {

public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

      }
      return null;
   }
}
```

```java
public interface Shape {
   void draw();
}
```

```java
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

```java
ShapeFactory shapeFactory = new ShapeFactory();

//get an object of Circle and call its draw method.
Shape shape1 = shapeFactory.getShape("CIRCLE");

//call draw method of Circle
shape1.draw();

//get an object of Rectangle and call its draw
method.
Shape shape2 = shapeFactory.getShape("RECTANGLE");
//call draw method of Rectangle
shape2.draw();
```

```java
public class Square implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Quelques Pattern

Proxy Pattern

- C'est un pattern permettant de faire des optimisations sur les accès aux objets.
- C'est la base de Aop

- Lorsque nous voulons une version simplifiée d'un objet complexe ou lourd. Dans ce cas, nous pouvons le représenter avec un objet Proxy qui charge l'objet d'origine à la demande, et fait initialisation différée. Ceci est connu comme le proxy virtuel
- Lorsque l'objet d'origine est présent dans un espace d'adressage différent et que nous souhaitons le représenter localement. Nous pouvons créer un proxy qui effectue tout ce qui est nécessaire comme la création et la maintenance de la connexion, l’encodage, le décodage, etc., pendant que le client y accède tel qu’il était présent dans leur espace adresse local. Ceci s'appelle le Remote Proxy
- Lorsque nous voulons ajouter une couche de sécurité à l'objet sous-jacent d'origine pour fournir un accès contrôlé basé sur les droits d'accès du client. Ceci s'appelle Protection Proxy

Proxy Pattern pour l’optimisation

```java
public interface ExpensiveObject {
    void process();
}
```

```java
public class ExpensiveObjectProxy implements
ExpensiveObject {
    private ExpensiveObject object;

    @Override
    public void process() {
        if (object == null) {
            object = new ExpensiveObjectImpl();
        }
        object.process();
    }
}
```

```java
public class ExpensiveObjectImpl implements ExpensiveObject {

    public ExpensiveObjectImpl() {
        heavyInitialConfiguration();
    }

    @Override
    public void process() {
        LOG.info("processing complete.");
    }

    private void heavyInitialConfiguration() {
        LOG.info("Loading initial configuration...");
    }

}
```

Proxy Pattern pour la sécurité

```
public interface ExpenseService {
    void processPaiement();
}
```

```
public class ExpensiveObjectProxy implements
ExpensiveObject {
    private ExpensiveObject object;
    private User user;
    public ExpensiveObjectProxy(ExpensiveObject
object){
this.object=object;}
    @Override
    public void processPaiement() {
        if (user.isAdministrator())
        object.process();
    }
}
```

```
public class ExpenseServiceImpl implements
ExpensiveObject {

    public ExpenseServiceImpl() {
        ();
    }

    @Override
    public void processPaiement() {
        LOG.info("Effectue le paiement.");
    }

}
```

Quelques pattern

Singleton Pattern

- Le singleton pattern est une solution pour avoir 1 seule instance d’une classe
- Comprendre quand l’utiliser et surtout quand ne pas l’utiliser.

Singleton Pattern

- Le modèle de conception singleton résout des problèmes tels que:
  - Comment peut-on être sûr qu'une classe n'a qu'une seule instance?
  - Comment accéder facilement à l'unique instance d'une classe?
  - Comment une classe peut-elle contrôler son instanciation?
  - Comment limiter le nombre d'instances d'une classe?
- Le design pattern est tout indiqué pour implémenter des services qui :
  - sont fonctionnellement uniques au sein de l'application (ex: système de logging centralisé, gestion de la configuration...)
  - doivent pouvoir être appelés par toutes les couches de l'application. Il serait en effet peu pratique de passer une référence au service à toutes les classes devant l'utiliser.
- Le modèle de conception singleton décrit comment résoudre ces problèmes:
  - Cache le constructeur de la classe.
  - Définissez une opération statique publique (getInstance ()) qui renvoie l'instance unique de la classe.

Singleton

```
public class Singleton
{
    /** Constructeur privé */
    private Singleton()
    {}

    /** Instance unique non préinitialisée */
    private static Singleton INSTANCE = null;

    /** Point d'accès pour l'instance unique du singleton */
    public static synchronized Singleton getInstance()
    {
        if (INSTANCE == null)
        {   INSTANCE = new Singleton();
        }
        return INSTANCE;
    }
}
```

Introduction à la programmation avec Java

- Les fondements de la programmation
- Les constructions de base d'un programmes
- Tests Logiciels
- Bonne Pratiques de conception d'une application
- Les techniques Objets
- Introduction à SVN
- Définition de la structure d'un projet avec Maven
- Quelques Design Pattern
- Mesure de la qualité

Les approches du test aujourd'hui

La notion de couverture des tests.

- Qu’est-ce qu’un test structurel? Fonctionnel?
- Comment trouver les bons tests structurels?
- Comment calculer la couverture d’un test?

Les types de tests

Cycle de vie

Bilan

Recette non fonctionnelle

Recette fonctionnelle

Recette technique

Réalisation

Conception

Etude d'impact

Etude d'opportunité

Test en Boite Blanche

Test en Boite Noire

Niveau d'accessibilité

Test fonctionnel

Test Performance

Test d'integration

Test Sureté

Test Ergonomie

Test de Sécurité

Test de conformité

.....

Caractéritique testé

Les types de tests

- Il y a trois grands types de tests par version de produit:
  - Les tests dit structurelles appelé souvent test en boite blanche car unitaire et technique.
  - Les tests fonctionnels exécuté en en boite noir
  - Les tests non fonctionnels. Il s'agit de test de performance, robustesse…

- Ces trois types de tests sont complémentaires et vérifie respectivement qu'un logiciel ne contient pas d'erreur, fait ce qu'il doit faire, et le fait « correctement »

- Enfin les tests dit de non régression devant montrer que ce qui fonctionnait avant…. fonctionne encore

Tests Structurels

- Les tests structurels s’appuie sur le code de l’application (d’où le surnom de test en boite blanche) pour faire ressortir des cas de tests.
- Ces cas de tests sont censé aboutir à une bonne couverture du code en terme de :
  - Nombre de chemin couvert
  - Nombre de débranchement couvert
  - Nombre d’instruction couvert.
  - Espace des données couvert
- L’hypothèse forte est que si un code est couvert à 100% alors il ne contient pas d’erreur. (A ne pas confondre avec ce code fait ce qu’il doit faire)

Tests Structurels

```
public Integer Sum(Integer x, Integer y)
{ int xx=x==null?0:x;
Int yy=y==null?0:y;
return xx+yy;
}
```

- L'espace de données est l'espace des entiers + la valeur null (Integer est objet).
- L'ensemble des chemins est égal à l'ensemble des débranchements
- Les cas de test critiques sont (null,0),(0,null),(null,null).

Tests structurels

- Si les tests structurels sont pratiqués de façon « parfaite » alors ils relèvent la totalité des erreurs structurelles.
- Cette méthodologie est très pratiquée dans le contexte de code critique (additionner avec des méthodes de preuves de programmes et de langage fortement type).
- Dans les autres contextes se sont des tests complexes à mettre en place et n'apportant pas de plus-value « fonctionnel ». Autrement dit, on test ce qui fonctionne et moins ce qui ne devrait pas fonctionner.
- La question est : « à la lecture du code, où dois-je placer des tests structurels ».
- L'utilisation de sonar est essentiel pour mettre en place des tests de structures

Tests structurels

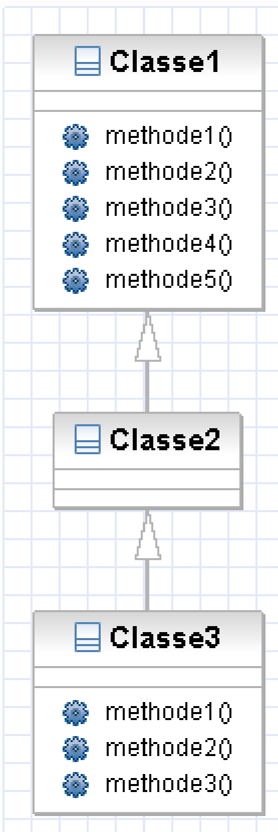
- Différents indicateurs de méthodes à tester structurellement :
  - Les triviaux: Nombre total de lignes de codes, Nombre de méthodes par objet, Nombre de méthode appelé dans une méthode…
  - Les indices de spécialisation, d'instabilité
  - La complexité cyclomatique
- Ces métriques sont des « trucs » pour placer au mieux les tests structurels.
- Un bon emplacement est aussi de tester de façon importante les parties désignées comme critique par les développeurs, les méthodes ayant dépassé le temps assigné pour les écrire, les méthodes ayant été « repris » en cours d'écriture par un développeur

Les indices de spécialisation en OO

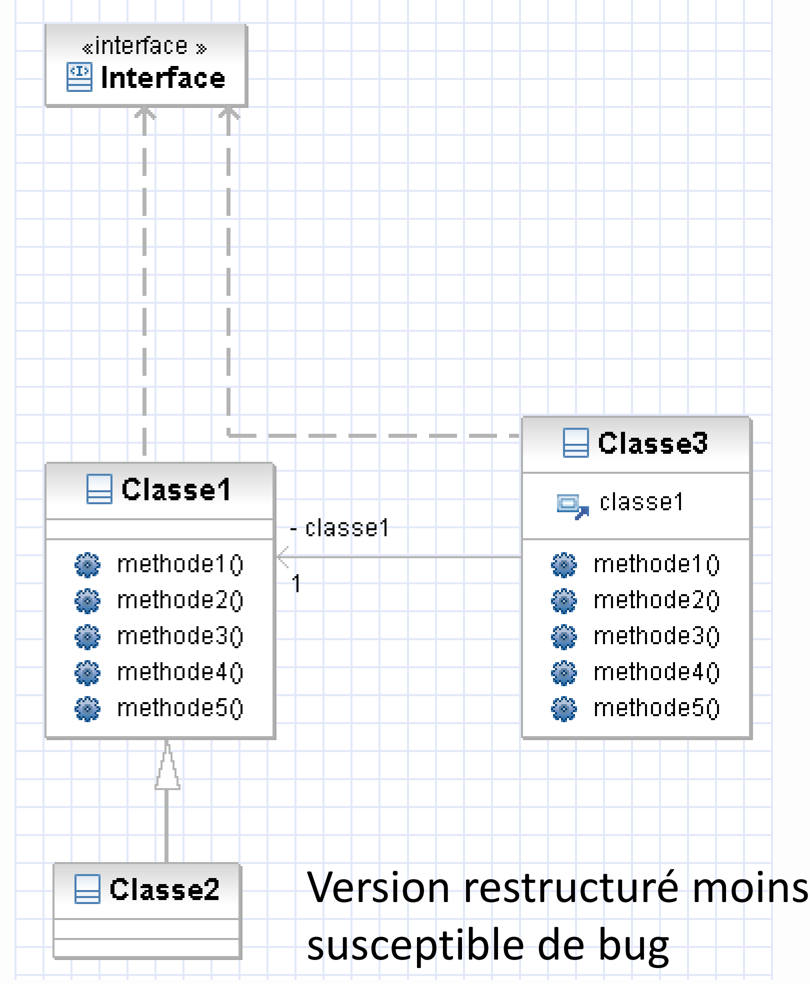
- L’indice de spécialisation se définit comme (NORM*DIT)/NOM.
  - NORM : Number of Overriden Methods, le nombre de méthodes redéfinies.
  - DIT : Depth in Inheritance Tree, la profondeur dans l'arbre d'héritage.
  - NOM : Number Of Methods, le nombre de méthodes de la classe étudié.

- L’indice est calculé pour toutes les classes d’un package.
- Les classes ayant un indices >1.5 sont de bonne candidates pour des tests structurels (trop de redéfinition de méthode pour une classe « trop » loin de la classe de basse)

Les indices de spécialisation en OO



L'indice de spécialisation est de 3 pour la classe 3 (3 méthode réécrite pour 3 méthode à 3 de distance)



Version restructuré moins susceptible de bug

Indice de stabilité en OO

- La stabilité d’une classe définit son côté « central » dans une application.
- Cet indice vaut Ce/(Ce+Ca) avec:
  - Ca : Couplage affèrent, le nombre de classes en dehors du paquetage qui dépendent de classes de ce paquetage.
  - Ce : Couplage efférent, le nombre de classes de ce paquetage qui dépendent de classes en dehors de ce paquetage.
- Cet indice va faire ressortir les paquetages qui dépendent plus des autres que les autres ne dépendent d'eux. Ces paquetages peuvent être dangereux, puisqu'une modification dans un des paquetages dont ils dépendent impacte potentiellement leur fonctionnement.
- Les tests structurels doivent donc d’abord s’attaquer au classe stable

Complexité cyclomatique

- La complexité cyclomatique d'une méthode est définie par le nombre de chemins linéairement indépendants qu'il est possible d'emprunter dans cette méthode.
- Plus simplement, il s'agit du nombre de points de décision de la méthode (if, case, while, ...) + 1 (le chemin principal).
- La complexité cyclomatique d'une méthode vaut au minimum 1, puisqu'il y a toujours au moins un chemin.

Complexité cyclomatique

- La complexité cyclomatique d'une méthode augmente proportionnellement au nombre de points de décision. Une méthode avec une haute complexité cyclomatique est plus difficile à comprendre et à maintenir.
- Une complexité cyclomatique trop élevée (supérieure à 30) indique qu'il faut restructurer la méthode (code non testable).
- Une complexité cyclomatique inférieure à 30 peut être acceptable si la méthode est suffisamment testée.

Outils

- Les outils de calculs des complexité sont nombreux mais difficile a analyzer pour les néophites.

- Quelques astuces de bon sens:
  - Eviter les méthodes longues ou trop complexe
  - Eviter de redéfinir trop de méthodes dans un héritages.
  - Tester en priorité les classes centrals de l’application

Outils

- Présentation de quelques outils simples de qualité de code.
- Découverte de Sonarlint et CheckStyle

SonarLint

- SonarLint est un outil permettant de vérifier le code source statiquement afin vérifier des bonnes pratiques.
- Il s’incorpore à Eclipse via un plugin. Ensuite il est possible de le configurer

Configuration SonarLint

Preferences

type filter text

- Cloud Foundry
- Code Recommenders
- Data Management
- Docker
- Forge
- Fuse Tooling
- Gradle
- Help
- HQL editor
- Install/Update
- Java
- Java EE
- Java Persistence
- JavaScript
- JBoss Tools
- JDT Weaving
- JSON
- JVM Monitor
- Language Servers
- Maven
- Metrics Preferences
- Mylyn
- Oomph
- Plug-in Development
- Project Archives
- Remote Systems
- Run/Debug
- Server
- SonarLint
  - Analyzer Properties
  - File Exclusions
  - Miscellaneous
  - Rules Configuration
- Spring
- Team
- Terminal
- Validation
- Visualiser
- Web
- Web Services
- XML
- YEdit Preferences

Rules Configuration

Configure rules used for SonarLint analysis. When a project is connected to a SonarQube/SonarCloud server, configuration from the server applies.

All rules

type filter text or rule key

- HTML
- Java
  - [x] ".equals()" should not be used to test the values of "Atomic" classes
  - [x] "=+" should not be used instead of "+="
  - [ ] "==" and "!=" should not be used when "equals" is overridden
  - [x] "@CheckForNull" or "@Nullable" should not be used on primitive types
  - [x] "@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects
  - [x] "@Deprecated" code should not be used
  - [ ] "@EnableAutoConfiguration" should be fine-tuned
  - [ ] "@Import"s should be preferred to "@ComponentScan"s
  - [x] "@NonNull" values should not be set to null
  - [x] "@Override" should be used on overriding and implementing methods
  - [x] "@RequestMapping" methods should be "public"
  - [x] "@RequestMapping" methods should specify HTTP method
  - [x] "@SpringBootApplication" and "@ComponentScan" should not be used in the default package
  - [x] "Arrays.stream" should be used for primitive arrays

"==" and "!=" should not be used when "equals" is overridden (squid:S1698)

Code smell   Minor

It is equivalent to use the equality `==` operator and the `equals` method to compare two objects if the `equals` method inherited from `Object` has not been overridden. In this case both checks compare the object references.

But as soon as `equals` is overridden, two objects not having the same reference but having the same value can be equal. This rule spots suspicious uses of `==` and `!=` operators on objects whose `equals` methods are overridden.

Noncompliant Code Example

```
String firstName = getFirstName(); // String overrides equals
String lastName = getLastName();

if (firstName == lastName) { ... }; // Non-compliant; false even if the strings have the same value
```

Compliant Solution

```
String firstName = getFirstName();
String lastName = getLastName();
```

Restore Defaults   Apply

Apply and Close   Cancel

SonarLint

- Exemple de règle a activer (ou pas).
- Lors de la création de l'opération equals pour un objet, le fait d'utiliser l'opérateur == pour ces objets est surement un bug:
  - Sinon pourquoi avoir créer cet opérateur.
  - Cela n'est pas un bug lorsque l'on souhaite faire une égalité de pointeur

SonarLint

- Exemple de bug révéler par SonarLint sur d’un code pour avoir une connexion à la base de données.
- Les connexions doivent être fermé en fin d’usage.
- Ici le bug est dans la partie Exception qui ne ferment pas la connexion

```
public Connection getConnection() {
        try {
            final java.sql.Connection connexion = this.ds.getConnection();
            final Configuration configuration = new
DefaultConfiguration().set(connexion).set(SQLDialect.POSTGRES_9_5);
            return new Connection(connexion, configuration);
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
```

PMD

- PMD est capable de détecter les failles ou les failles possibles dans le code source, comme:
  - Bugs possibles - Videz les blocs try / catch / finally / switch.
  - Code mort: variables locales, paramètres et méthodes privées non utilisés.
  - Vide les déclarations if / while.
  - Expressions surchargées - Inutiles si instructions, pour les boucles qui pourraient être des boucles while.
  - Code sous-optimal: utilisation inutile de String / StringBuffer.
  - Classes avec mesures de complexité cyclomatique élevée.
  - Code dupliqué - Le code copié / collé peut signifier des bogues copiés / collés et réduit la facilité de maintenance.
- Un plugin Eclipse officiel est disponible

PMD

- 2 bugs de commentaires sont vue. Il n’y a pas de commentaires sur des méthodes publiques
- Le nom des paramètres sont trop court et peu parlant.
- Une méthode ne devrait avoir qu’un seul return

```
public boolean isUUID(final String s) {
        return this.fromString(s) != null;
    }

    public UUID fromString(final String s) {
        if (StringUtils.isNoneEmpty(s)) {
            try {
                return UUID.fromString(s);
            }
            catch (IllegalArgumentException e) {
                return null;
            }
        }
        return null;
    }
```