**Formation Java SE - Titre pro**
Formateur : Mohamed DERKAOUI (mderkaoui@dawan.fr)

Notes : https://mensuel.framapad.org/p/java-11012021

Support slides formateur :
http://demo.dawan.biz/mohamed/java1.pdf
http://demo.dawan.biz/mohamed/java2.pdf

Tutoriel Oracle : https://docs.oracle.com/javase/tutorial/

JavaDoc 1.8: https://docs.oracle.com/javase/8/docs/api/

Projet du formateur : http://demo.dawan.biz/mohamed/project1.zip

Messagerie : https://mail.dawan.fr
Intranet : https://dawan.org

---

Install : jdk puis Eclipse STS
Participants :

- Erwan : suit formation web
- Valentin : Absent vendredi

- Simon :
- Tanguy :
- Ahmed :
- Maxime :
- Alfoussein :
- Yosra :
- Jade:
- Sylvain :
- Denis :
- Fabien :
- Kanha :
- Laura :
- Aser :
- Nicolas :
- Romain :
- Ali :

http://www.java2s.com/Code/JarDownload/java-json/java-json.jar.zip
Dézipper l'archive
Créer dans le projet un répertoire lib, copier le ".jar" dedans
Puis click droit sur le jar - > build path - > "Add to Build Path"

Javadoc : https://stleary.github.io/JSON-java/org/json/JSONWriter.html

http://demo.dawan.biz/mohamed/SerialTools.java
http://demo.dawan.biz/mohamed/FilesTest.java

---

**Styles de programmation :**

- impérative : suite d'instructions dans un prog
  inconv. : pas de réutilisation de code
- procédural : regroupement en fonctions
  av. : réutilisation
  inc. : difficulté à représenter des objets
- orientée objet : représentation d'objets à l'aide de classes
  av. : modularité
  inc. : lourdeur et difficulté d'architecture

---

**Découvrir la plateforme Java**

Historique, versions
Langage de programmation 100% objets
Indépendant de la plateforme (pas besoin de recompiler le code pour l'exécuter sur d'autres OS)
Créé dans les années 90 (Sun Micro > Open Source > Oracle)

---

**Editions Java :**

- Java SE : Standard Edition (biblio de bases, IHM client lourd)
- Java EE : Entreprise Edition (composants web)
- Java ME : Mobile Edition (composants embarqués)

---

**Compilation et interprétation par la JVM (Java Virtual Machine) :**

source.java ===compilation (javac.exe) ====> .class (bytecode : code intermédiaire)
======interprétation par la JVM (java.exe)=> code machine

Exemple :
javac.exe Contact.java =======> Contact.class

java.exe Contact

la JVM (machine virtuelle Java) est dépendante de l'OS
Le code Java est indépendant de l'os (pas besoin de le compiler)

---

**Technologies/frameworks Java et positionnement**

Généralement, une application en couches :

- Présentation :
  * Client lourd : Awt/Swing, JavaFx
  * Client léger (web) :
    composants de base : Servlet / JSP
    frameworks MVC : Struts, JSF, Spring Web MVC
- Métier :
  classes ou bibliothèques de classes
  Web services :
    REST => API JAX-RS
    SOAP => API JAX-WS
- Accès aux données :
  classes
  bibliothèques de classes
  web services

  NB. :
  JDBC (Java Database Connectivity) : api permettant de se connecter aux Bdds relationnelles (équivant à ADO.NET)
  JPA (Java Persistence API) : spécification des interfaces nécessaires pour l'implémentation d'un ORM
    Implémentation de JPA : Hibernate, EclipseLink, iBatis, OpenJPA, ....

Remarque : il existe des outils de build (construction du code) permettant de gérer les dépendances d'un projet
Ant, "Maven", Gradle,...

---

**Environnement de développement :**

- JDK : Kit de développement Java (outils / JRE)
- IDE : Eclipse, NetBeans, IntelliJ

---

**Empaquetage et déploiement d'une application Java :**

- client lourd ou bibliothèques de classes : .jar (Java ARchive)
- client léger ou web service : .war (Web ARchive)
- .ear(Entreprise ARchive) : enveloppe avec plusieurs war et plusieurs Jars

Pour exécuter une application web, on a besoin d'un serveur d'applications Java EE :

Apache Tomcat
Oracle GlassFish
Redhat JBoss
IBM WebSphere

Oracle WebLogic
....

---

Installation du JDK :
.exe à exécuter
Optionnellement, on peut définir une variable d'environnement : JAVA_HOME

Installation d'un IDE et paramétrage :
Encodage : Window > Preferences > General > Workspace
changer l'encodage de windows (CP1252) en unicode (UTF-8)

JRE : Window > Preferences > Java Installed JRE
Add > Standard VM > pointer sur le répertoire du JDK installé
puis cocher la case correspondante

Thème : Window > Preferences > Appearance

---

Package (Namespace ) : regroupement logique et physique (répertoire) de plusieurs fichiers
Intérêt : organiser le code et éviter des conflits de noms

Un fichier .java porte le nom de la classe "public"
1 classe "public" par fichier.java

---

**Maîtriser les bases**

Utilisation de variables, constantes, opérateurs

**Variable** : type nom1, nom2, nom3 = valeur;

**Constantes** : mot clé "final"
final type NOM_CONSTANTE = valeur;

**Opérateurs** :
arithmétiques : +, -, *, /, % (modulo)
++, --
combinés : +=, -=, *=, /=, %=
a = a + b; <=> a += b;

de comparaison : >, <, >=, <=, ==, !=
logiques : &&, ||, ! (not), ^ (xor)

**Types simples et types références** :
types simples ont des cases alloués dans la pile (stack)
types références ont des pointeurs (stack) qui référencent des objets dans le tas (heap)
(les types références sont des types objets)

**Transtypage, Wrappers** :
voir les wrappers et les méthodes parseXXXX

**Expression de conditions :** if/else, switch, opérateur ternaire
if(condition) { .... }
else { ...}

**Utilisation de boucles** : for, while, do while
for : connaissance du nombre d'itérations
for (prononcée foreach) : parcours complet d'un tableau ou d'une collection
while : boucle conditionnelle
do while : boucle conditionnelle exécutée au moins une fois (faire .... tant que (condition))

Manipulation de tableaux :
- 1 seule dimension : type[] nom = new type[taille];
- 2 dimensions : type[][] nom = new type[nbLignes][nbColonnes];
- tableau de tableaux : type[][] nom = new type[nbCases][];

---

**Factorisation de codes avec méthodes :**
visibilité mot-clé typeRetour nomMethode(paramètres)
Une méthode peut remonter des exceptions avec le mot clé "throws"

---

**Surcharge (overload) :**
Plusieurs méthodes peuvent avoir le même nom et différents paramètres (nombres/types)
int sum(int a, int b)
double sum(double a, double b)
int sum(int a, int b, int c)

---

**Arguments variables :**
type... nomParam
1 seul var-arg par méthode

```
//MyMethods.multiply(12, 2);
    //MyMethods.multiply(12, 2,56);
    //MyMethods.multiply(tab1);
    public static double multiply(double... t) {
        double res = 1;
        for (double d : t) {
            res*=d;
        }
        return res;
    }
```

---

Un algorithme peut être :
- soit itératif : en utilisant des boucles
- soit récursif : en rappelant la méthode à l'intérieur de son corps

**Récursivité :** pouvoir appeler la même méthode avec différents paramètres

n! = n * (n-1)!

```
public static int fact(int n){
    if(n<=1) return 1;
    return n * fact(n-1);
}
```

**Exo 3 : écrire une méthode prodChiffres qui multiplie les chiffres d'un nombre entier "n**

prodChiffres(254) = 4*5*2

254%10 = 4

254/10 = 25

prodChiffres(9) = 9

prodChifres(254) = 4 * prodChiffres(25)

---

**Commenter et documenter du code**

- commentaire non technique : // ou /* */
- commentaire pour la javadoc (doc technique) : /** */
  avec utilisation d'annotations @version, @author, @param,....
  ** on pourra générer les docs html avec l'outil : javadoc.exe
  depuis Eclipse : menu Project > Generate Javadoc

---

**Enumération** :

ensemble de constantes
peut être définie au sein d'une classe
ou être associée à un package
peut être définie dans son propre fichier Java

---

**Apprendre l'objet**

Définition de classes, POJO vs JavaBean

- Objet : élément identifiable (abstrait ou concret)
- Classe : Définition d'un type d'objets

Elle peut contenir :

- des variables (des attributs)
- des méthodes
- des constructeurs

Toute classe Java hérite de la classe "Object", elle récupère un ensemble de méthodes qu'on peut redéfinir

---

**//Exo : Ecrire une classe CompteB (numero, solde, depot(montant), retrait(montant))**

**//depot et retrait peuvent déclencher une exception si solde insuffisant pour le retrait**
**//ou montant incorrect**
**// Appeler les méthodes associées dans le main**

Correction :

http://demo.dawan.biz/mohamed/BankAccount.java
http://demo.dawan.biz/mohamed/OOPTest.java

---

**Déclaration des membres d'instance :**

variables propres à chaque objet
numéro et solde sont des variables propres à chaque Compte bancaire

**Variables de classe (static) :** partagées par toutes les instances
pas besoin d'instancier la classe pour utiliser les variables
Les variables "static" peuvent initialisées au moment de la déclaration ou dans un bloc static { ... }

**Constructeurs et instanciation :**

Constructeur : méthode spéciale qui porte le même nom que la classe et qui permet d'initialiser les attributs.

Visibilité NomClasse(....) { ..... }

Instanciation :

NomClasse nomObjet = new Constructeur(....);

**Facteurs de visibilité (modificateurs d'accès) :**

public : visibilité globale
private : visibilité classe
protected : visibilité classe et sous-classes
(rien : default) : visibilité package

---

**Cycle de vie d'un objet en mémoire :**

Un objet est alloué dans le tas avec un pointeur qui conserve son adresse.
Quand le pointeur est mis à null, l'objet n'est pas détruit immédiatement (gestion auto). Il existe un système de ramasse miettes (garbage collector) qui se chargera de la destruction de l'objet non référencé
On peut redéfinir (@Override) la méthode finalize afin de réaliser un traitement avant le destruction de l'objet.

---

**Diagramme de classes (UML) :**

UML : Unified Modeling Language
il propose 13 diagrammes.
Il existe un diagramme statique : "Diagramme de classes"

plugin objectAid UML : http://www.objectaid.com/update/current

---

**Encapsulation : getters et setters**

le fait de protéger l'accès aux attributs afin d'ajouter du contrôle
Procédé : rendre les attributs private et ajouter des méthodes d'accès : getters/setters

---

**Agrégation d'objets (association) :** le fait d'inclure un objet dans un autre

Un Client possède une adresse

Une voiture possède un moteur

Il existe 2 types d'association :

- agrégation (faible) : l'objet interne n'est pas obligatoire
- composition (forte) : l'objet interne est obligatoire

NB. : ne pas oublier d'instancier l'objet interne.

si agrég forte : instanciation dans le constructeur

sinon instanciation avant utilisation

Correction Exo Qcm :

http://demo.dawan.biz/mohamed/Quiz.java

http://demo.dawan.biz/mohamed/QuizQuestion.java

http://demo.dawan.biz/mohamed/QuizResponse.java

http://demo.dawan.biz/mohamed/AgregationTest.java

---

**Extension de classes (Héritage) :**

- Etendre une classe existante
- Une classe fille n'hérite que d'une seule classe mère (pas d'héritage multiple en Java)
- Une classe fille hérite de l'ensemble des membres (attributs et méthodes) de la classe mère mais ne peut accéder qu'aux membres public ou protected
- Une classe fille n'hérite des constructeurs de la classe mère mais peut les appeler grâce au mot-clé super(...)
- Une classe fille peut redéfinir (@Override) une méthode héritée à condition qu'elle ne soit pas marquée "final"

Exemple :

CompteB(-numero, -solde, +getSolde())

CompteEp extends CompteB

- Toute classe hérite implicitement d'Object

On peut redéfinir les méthodes héritées :

* toString() => représentation en chaine de caractères de l'objet
* equals(...) => comparaison d'objets
* hashCode()=> un entier représentant le code de hachage
* finalize => méthode appelée à la destruction de l'objet
* clone() => cloner un objet

---

Comparaison d'objets :

On peut comparer des objets en utilisant :

- l'opérateur == : comparera uniquement les pointeurs
- la méthode equals de la classe Object => boolean. La méthode doit être redéfinie dans la classe concernée
- la méthode compareTo de l'interface Comparable
- la méthode compare de l'interface Comparator

---

**Classe abstraite (abstract) :**

- non instanciable
- définit un modèle pour les sous-classes
- peut contenir :
  * des attributs
  * des méthodes
  * des constructeurs (appelés par les classes filles grâce au mot-clé "super")
  * des méthodes abstraites : signatures de méthodes que les sous-classes concrètes doivent obligatoirement définir.

Exemple : oop.abstraction

abstract class TrainingDoc (title, author, constructeurs, abstract void Print();)

class Book extends TrainingDoc

---

**Interfaces et implémentation**

Interface = contrat qui définit un ensemble de méthodes que les classes concrètes ont l'obligation de définir

```
//qlq de chose est Pliable s'il possède les 2 méthodes : plier et deplier
public interface Pliable {
    void plier();
    void deplier();
}
```

Une classe peut implémenter une ou plusieurs interfaces :

```
public class Chaise implements Pliable {
    private String modele;

    public void plier(){
        .......
    }

    public void deplier(){
        .......
    }
}
```

---

Implémenter l'interface Cloneable dans la classe Customer ; redéfinir clone

Ensuite tester si le clone de l'objet Customer contient bien des objets agrégés séparés

Correction :

http://demo.dawan.biz/mohamed/Customer.java

http://demo.dawan.biz/mohamed/Address.java

http://demo.dawan.biz/mohamed/OOPTest.java

---

**Polymorphisme**

Un objet peut prendre plusieurs formes

On peut le typer avec :

- soit une classe mère
- soit une interface

Intérêt :
Manipuler un objet sans se soucier de son type
Mélanger des objets de différentes natures

```
Pliable p = new Chaise();
Pliable p2 = new Table();

BankAccount ba = new SavingAccount();
TrainingDoc d = new Book(....);

public void acheter(Pliable ch){
   ...
}
```

---

**Gérer les exceptions**

Définition :

Exception = évènement qui arrête l'exécution
On peut :
- soit le capturer et le traiter : try/catch/finally
- soit le remonter pour le traiter au moment de l'appel throw/throws

La classe Throwable est la classe mère de toutes les Exceptions et les erreurs

Throwable
Error Exception

Types d'exceptions :

Il existe 2 types d'exceptions : contrôlées / non contrôlées

* les exceptions contrôlées (vérifiées) :
héritent de java.lang.Exception
pour une condition récupérable, telles que les erreurs de compilation
déclaration explicite de try/catch

* les exceptions non contrôlées :
héritent de java.lang.RuntimeException
pour les conditions irrécupérables (erreurs d'exécution)
aucun try/catch n'est explicite

Capturer et traiter une exception (try/catch/finally) :

```
try{ //    essayes
   //code susceptible de déclencher une exception
}catch(TypeException e){
   //traitement
}catch(Type2Exception e){
```

```
  //traitement
}
```
On peut réaliser le même traitement pour différents types d'exceptions
```
try{
   .....
} catch(TypeException1 | TypeException2 e) {
   ...
}
```

Le bloc finally est exécutée que l'exception soit levée ou pas
```
try//essayes
catch//si tu attrapes une exception de type XXX
finally//et au final, faire ....
```

Lever/Remonter une exception (throw/throws) :

Dans une méthode, on peut choisir de traiter toute de suite l'exception avec un try/catch ou de la remonter pour la traiter au moment de l'appel

```
public static double divide(double a, double b) throws Exception {
     if(b==0)
          throw new Exception("Erreur : le dénominateur ne peut être nul");
```
- `return a/b;`

```
}
```

**Création d'exceptions :**

On peut créer nos propres exceptions en héritant d'Exception ou une de ses dérivées
Exemple :
```
    class AgeInvalidException extends Exception{
        //....
    }
```

---

**Utiliser des collections**

Structure de données dynamique (taille variable)
Il existe 2 types de collections : génériques (typées), non génériques

Présentation de l'API disponible, generics :

le package java.util contient un ensemble d'interfaces et de classes
(I) List, (I) Set, (I) Map,...
    (C) ArrayList<T>, (c) Vector<T>,....
voir la javadoc : https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html

Stack : pile, LIFO
Queue : file, FIFO

List : indexée
Set : ensemble sans doublon (peut être ordonné)
Map : ensemble de clés/valeurs
....

Exo Panier :
http://demo.dawan.biz/mohamed/Cart.java
http://demo.dawan.biz/mohamed/CartLine.java
http://demo.dawan.biz/mohamed/Product.java
http://demo.dawan.biz/mohamed/CartTest.java

**Comparatif, choix d'un type de collection :**
Le choix dépend des éléments : null, doublons
ainsi que des opérations à réaliser :
Différences entre les collections : https://en.wikiversity.org/wiki/Java_Collections_Overview

Classes essentielles : ArrayList, HashMap,...
voir la javadoc : https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html

Parcours, opérations sur des collections et tris

- Tri : Collections.sort(...), on peut utiliser des comparateurs
- (voir exemple)

---

**Manipuler des fichiers**

package java.io
package java.nio

classes : File, Files,...

On peut manipuler des fichiers :
au format binaire : InputStream, OutputStream
au format caractères : Reader, Writer

Lecture et écriture de fichiers :
FileInputStream / FileOutputStream
FileReader / FileWriter

Utilisation de buffers :
BufferedInputStream / BufferedOutputStream
BufferedReader / BufferedWriter

Manipulation de chemins : Path, File
Répertoires : File

http://demo.dawan.biz/mohamed/CsvTools.java
http://demo.dawan.biz/mohamed/FilesTest.java

---

==========

Ex :
Votre client vous demande de développer une nouvelle application dont les paramètres techniques sont fournis dans un fichier de configuration yaml.

Pour rappel, un fichier yaml est un fichier sémantiquement proche du format XML
mais où chaque section est indentée de 4 espaces.
Le fichier de configuration qui vous sera fournie aura le format

```
database
    host: localhost
    user: root
    password: azerty
log
    file: /var/log/app.log
    level: debug
security
    cipher: qwerty
    algo: aes
```

Le but : écrire une fonction permettant de lire le fichier de configuration et de créer les instances d'objet correspondants.

- Etape1 : Créer les classes DatabaseConfiguration, LogConfiguration et SecurityConfiguration.
  Ces classes auront autant d'attribut que précisé dans le fichier de configuration :

    DatabaseConfiguration
      - String host
      - String user
      - String password

    LogConfiguration
      - String file
      - String level

    SecurityConfiguration
      - String cipher
      - String algo

- Etape 2 : lire le fichier de configuration et renvoyer la configuration correspondante.

```
public class ConfigurationParser {
```

```
    public static Configuration parse(String file) {
        // TODO
    }

    public static class Configuration {
        public DatabaseConfiguration database;
        public LogConfiguration log;
        public SecurityConfiguration security;
    }
}
```

Etape 3 : si ce n'est pas déjà fait, pouvez faire en sorte que l'ordre des sections soit aléatoire ?

Correction :
http://demo.dawan.biz/mohamed/ConfigurationParser.java
http://demo.dawan.biz/mohamed/FilesTest.java
==========
Spécification : https://yaml.org/spec/1.2/spec.html#id2764044

---

**Sérialisation d'objets :**

Le fait de stocker l'état d'un objet dans un support persistant (fichier, bdd,...)

Il existe plusieurs types de sérialisation :

- - binaire : ObjectOutputStream (pour désérialiser : ObjectInputStream)
- - XML : XMLEncoder / XMLDecoder pour désérialiser
- - JSON : JSONWriter / JSONTokener
- ...

---

**Externalisation de configuration dans des .properties**

Un fichier.properties contient un ensemble de clés/valeurs
On peut le manipuler grâce à la classe "Properties"

---

**Gestion des logs : java.util.logging, Log4j :**

En Java, vous pouvez :
- soit utiliser l'API native (java.util.logging)
- soit utiliser une api externe Log4j

Apports de Log4j : multiples supports de logs
On pourra écrire dans différents supports : File, Database, ....

**Vocabulaire :**

Logger : objet permettant d'écrire les logs (on pourra définir plusieurs loggers)
Appender : support de logs (FileAppender, RollingFileAppender, SmtpAppender,...)
Layout : Type de logs (Text, Xml)

Pattern : Format du message (date heure classe message saut de ligne)
Level : Niveau du message (info, warning, error, ...)

**Mise en place de Log4j :**

v1.x : ajouter log4j.jar + fichier de config : log4j.properties ou log4j.xml
v2.x : ajouter log4j-core.jar / log4j-api.jar + fichier de config : log4j2.xml

Exemple log4j2 :

http://demo.dawan.biz/mohamed/log4j-api-2.14.0.jar
http://demo.dawan.biz/mohamed/log4j-core-2.14.0.jar

+ ajouter dans src un fichier log4j2.xml :
http://demo.dawan.biz/mohamed/log4j2.xml

Exemple d'écriture de logs :

```
System.out.println("Log4j _____");
        Logger myLogger = LogManager.getRootLogger();
        myLogger.info("Ceci est mon premier log");

        Logger myErrorLogger = LogManager.getLogger("myErrorLogger");
        myErrorLogger.error("error on connection");
```

---

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
postgres
postgres

Téléchargement du driver + ajout dans /lib puis click droit > Build Path > Add to BuildPath
http://demo.dawan.biz/mohamed/postgresql-42.1.4.jar

____initialisation des données de la base en ligne de commande et démarrage confi/service_____

```
initdb.exe -E 'UTF-8' -D "C:\datapostgres" --username=postgres --auth=trust
pg_ctl.exe start -D "C:\datapostgres"
psql --username=postgres
ALTER USER postgres WITH PASSWORD 'postgres';
create extension adminpack;
```

---

NB. : Si vous redémarrez la machine, pensez à redémarrer le service :
se positionner dans : c:\Program Files\Posgre\11\bin\
pg_ctl.exe start -D "C:\datapostgres"

---

---

**Accéder à des bases de données**

Présentation de l'API JDBC :

JDBC : Java DataBase Connectivity (java.sql)
API pour se connecter à des bases de données relationnelles

Etapes :
- ajouter un driver (.jar) spécifique au SGBD utilisé
- établir une connexion :
  Class.forName("org.postgresql.Driver");
  String url = "jdbc:postgresql://localhost:5433/db1";
  String user="postgres";
  String pwd = "postgres";
  Connection cnx = DriverManager.getConnection(url,user,pwd);
- effectuer des requêtes :
  PreparedStatement ps = cnx.prepareStatement(requête SQL);
- exécuter la req :
  si SELECT => ResultSet rs = ps.executeQuery();
  parcours avec un itérateur puis fermeture
  Sinon => int nb = ps.executeUpdate();
- fermer la connexion

---

Pattern DAO (Data Access Object)
propose une organisation de classes afin de fournir des méthodes manipulant des objets et interagissant avec les tables en Bdd

(c) ProductDao

- List<Product> findAll()

List<Product> findAll(int start, int nb)
int insert(Product p)
....

Ecriture de requêtes et traitement des résultats :
http://demo.dawan.biz/mohamed/DbConnection.java
http://demo.dawan.biz/mohamed/JDBCTest.java
http://demo.dawan.biz/mohamed/ProductDao.java

**Gestion des transactions**

Transaction : ensemble de requêtes atomiques. Si l'une échoue, on revient à l'état initial

début transaction

- essayes
- req1
- req2
- commit
- si erreur
- rollback
- ----

Connection en JDBC est en auto-commit par défaut

- 

Exemple : https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

---

**Présentation de frameworks ORM**

JPA : Java Persistence API
spécification d'un ORM pour le mapping entre les objets et les tables
Plusieurs implémentations : Hibernate, EclipseLink, ...

---

**Notions avancées**

Optimiser du code, généricité

Généricité : concept permettant de créer des classes/méthodes/interfaces indépendantes du type d'objets
On peut ajouter des contraintes sur les generics en mettant <LeType extends XXX>
XXX peut être une classe ou une interface.

Introspection (Reflection API)

La réflexion permet d'introspecter un objet pour accéder à son type
Class cl = objet.getClass();
Class<T> permet d'avoir un type générique ; l'appel s'effectuera avec NomType.class.

Exemple:
http://demo.dawan.biz/mohamed/CsvToolsGen.java
http://demo.dawan.biz/mohamed/GenericTest.java

---

**Gestion des processus**

Processus : programme en exécution. Dans un processus, on aura moins 1 thread principal
Thread : tâche d'exécution
On peut créer plusieurs threads au sein d'un processus (multi-tâches)

Exemple 1 :
lancer des processus avec Process/ProcessBuilder ou Runtime
http://demo.dawan.biz/mohamed/TestProcessThreads.java

lancer une calculatrice, saisir dans la calculatrice 8
Process p écrire dans le p.getOutputStream().write("8".getBytes())

lancer cmd.exe ipconfig, récupérer le résultat
Process p p.getInputStream()

lancer cmd.exe, écrire dans le process ipconfig, récupérer le résultat
Process p , p.getOutputStream().write("ipconfig".getBytes()), p.getInputStream()

Exemple 2 :
classe Thread

---

**Options JVM :**

La JVM est une machine (process) qui interprète le bytecode généré par la compilation
Elle consomme des ressources mémoire et peut être paramétrée grâce à une variable d'environnement :
JAVA_OPTS
La liste des options de la JVM :
https://www.oracle.com/java/technologies/javase/vmoptions-jsp.html

JAVA_OPTS
-Xms1G -Xmx2G -XX:+DisableExplicitGC

En cas d'absence d'arguments de la JVM au sein du programme lancé, c'est les arguments de JAVA_OPTS qui seront pris en compte.

Afin de monitorer la JVM, un outil (jvisualvm)est disponible dans le répertoire du jdk/bin
c:/Program Files/Java/jdk_XXX/bin

---

**Gestion des Dates (API Time) :**

```
//java.util.Date
//Avant Java 8
Date date1 = new Date();
System.out.println(date1);

//Date=> String
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
System.out.println(sdf.format(date1));

//String => Date
try {
	Date date2 = sdf.parse("12/01/2020");
	System.out.println(date2);

	Calendar cl = Calendar.getInstance();
	cl.set(2021, 1, 1);
	Date date3 = cl.getTime();

} catch (ParseException e) {
	e.printStackTrace();
}

//Avec Java 8+
//API Time
//classes : LocalDate, LocalTime, LocalDateTime, Period, Instant,TimeZone...
LocalDate d4 = LocalDate.now();
```

```
        System.out.println(d4);

        LocalDate d5 = LocalDate.of(2021, 1, 15);

        LocalTime lTime = LocalTime.now();
        System.out.println(lTime);
```

---

**Expression Lambda**
**Interfaces fonctionnelles**
**Streams**

**Communications réseau : Sockets**

---

http://demo.dawan.biz/mohamed/MainWindow.java
http://demo.dawan.biz/mohamed/AddUpdWindow.java

---

Raccourcis Eclipse :
- indentation : ctrl+shift+F
- complétion : ctrl+espace
- comment/décommenter : ctrl+shift+/
- renommer : click droit refactor > rename
- organiser les imports : ctrl+shift+o
- vérifier le build path du projet : click droit > build path > configure buildpath > onglet librairies

---

---

Import d'un projet dans Eclipse :
- http://demo.dawan.biz/mohamed/project1.zip
- Décompresser l'archive