Java

CM4-4 : Java « avancé »

Mickaël Martin Nevot

V1.2.0

Java

Pile/tas

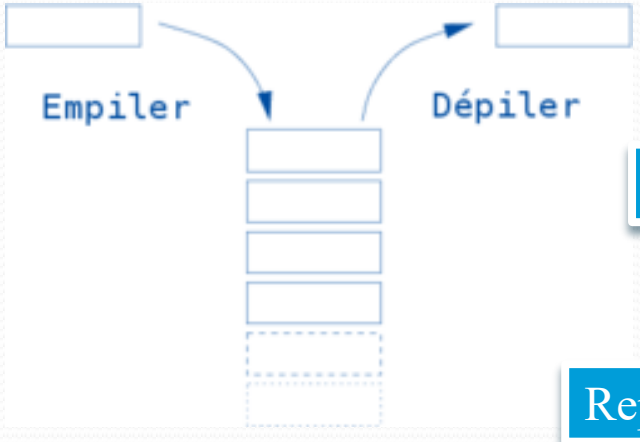
Fonctionnement dépendant de l'implémentation de la JVM : seul le principe général est expliqué ici !

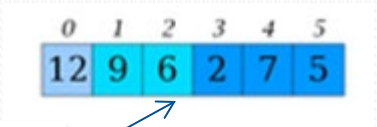
Pile (ou registre)

- Principe d'une pile :
  - Sommet : dernier élément
- Appels de méthodes
- Variables locales

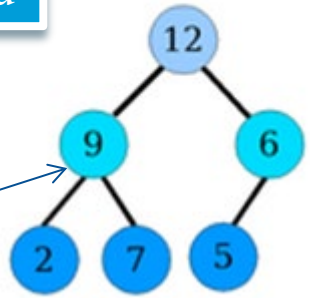
Tas

- Objet :
  - Place allouée par `new`
  - Contient attributs/méthodes
  - $taille_{objet} = \sum taille_{attributs}$
  - *Garbage collector*





Représentation en tableau



Représentation en arbre

Destruction et garbage collector

- Destruction implicite en Java
- *Garbage collector* (ou ramasse-miettes):
  - Appel automatique :
    - Si plus aucune variable ne référence l'objet
    - Si le bloc dans lequel l'objet est défini se termine
    - Si l'objet a été affecté à `null`
  - Appel manuel :
    - Instruction : `System.gc();`
- Usage de **pseudo-destructeur** :
  - Classe utilisateur : `protected void finalize()`
  - Appelée juste avant le *garbage collector* ? Pas certain !

Pour être sûr que `finalize()` soit appelée, il faut appeler manuellement le *garbage collector*

Flux

- Flux de données (comme un film en « streaming » !)
- Paquetage `java.io` :
  - Flux binaires (lecture/écriture d'octets) :
    - `InputStream`, etc.
    - `OutputStream`, etc.
  - Flux de caractères (lecture/écriture de caractères) :
    - `Reader` (`BufferedReader`, `FileReader`, etc.)
    - `Writer` (`BufferedWriter`, `FileWriter`, etc.)

```
// Lit l'entrée standard.
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
// Lit la ligne jusqu'au prochain retour chariot.
String inputLine = br.readLine();
```

Sérialisation

- Interface `Serialisable`
- Mot clef `transient` (pas de sérialisation) :

```
public class MyClass implements Serialisable {
    …
    protected MyClass2 myObj1 = new MyClass2();
    // myObj2 ne sera pas sérialisé.
    transient MyClass3  myObj2 = new MyClass3();
}
```

- « Désérialisation » :

```
FileInputStream fis = new FileInputStream ("myFile.ser");
ObjectInputStream ois = new ObjectInputStream (fis);
Object first = ois.readObject ();
MyClass myObj = (MyClass) first;
ois.close();
…
```

L'extension du fichier n'est pas obligatoire et n'a aucune importance

Collections (Java 2)

- Désavantages d'un tableau :
  - Taille statique
  - Recherche lente (exhaustive)
  - Pas de *pattern* de déplacement dans les éléments
- API Java :
  - Collection (interface `Collection`) :
    - Généricité et références (n'importe quelles références objets)
    - Opérations optimisées et communes
    - Itérateurs (parcourent les éléments un à un sans problème de type)
  - Tableau dynamique : `ArrayList`
  - Liste : `LinkedList`
  - Ensemble : `HashSet`, `TreeSet`

Itérateurs

- Monodirectionnel : interface `Iterator`
  - Toutes les collections en ont un :

```
// C'est une collection : on récupère son itérateur.
Iterator iter = c.iterator();
while (iter.hasNext()) {
    MyClass o = iter.next();
}
```

- Bidirectionnel : interface `ListIterator` (dérive de `Iterator`)
  - Listes et tableaux dynamiques uniquement
  - Deux sens

```
ListIterator iter = c.listIterator();
while (iter.hasPrevious()) {
    MyClass o = iter.previous();
}
```

Exemples de collections

- `LinkedList` (liste doublement chaînée)
- `ArrayList` (à la place de `Vector` qui est déprécié) :
  - Encapsulation du tableau avec une taille dynamique

```
ArrayList arrList <String>= new ArrayList<String>();
arrList.add("toto"); // Valide.
arrList.add(new String ("tata")); // Valide.
arrList.add(10); // Non valide ...
```

- `HashSet` :
  - Permet des éléments identiques
  - Prévoit la redéfinition des méthodes :
    - `hashCode()` : ordonnancer les éléments
    - `equals(...)`

Ellipse : varargs (Java 5)

- Nombre indéfini de valeurs de même type en paramètre
- Traitée comme un tableau
- Deux manières :
  - Avec un tableau (éventuellement vide)
  - Avec un ensemble de paramètres
- Placée en dernier dans la liste des paramètres
- En cas de surcharge de méthode, la méthode contenant l'ellipse a la **priorité la plus faible**

```
public meth1(Type... tab) { … }
…
int[] t = {1, 2, 3, 4, 5};
meth1(t); // Envoyé comme un tableau.
meth1(1, 2, 3, 4, 5); // Envoyé comme un ensemble de paramètres.
```

Utilisation de l'ellipse : …

JAR/WAR

- Formats de fichier
- JAR (extension `.jar`) :
  - Outil d'archivage du *bytecode* et des **métadonnées**
    - Fichier *manifest* : `MANIFEST.MF`

```
Manifest-Version: 1.0
Created-By: 1.4.1_01 (Sun Microsystems Inc.)
Main-class: HelloWorld
```

Classe principale à exécuter

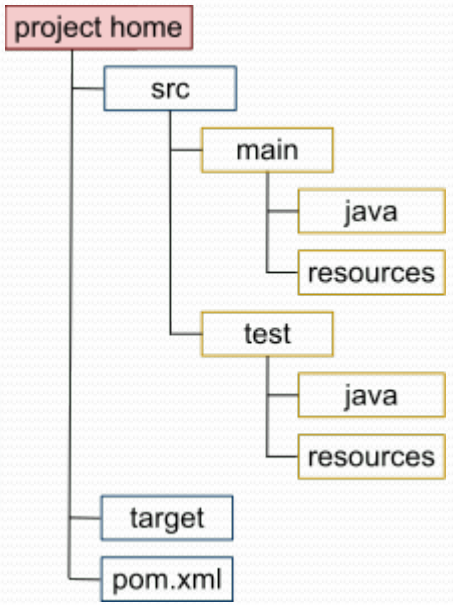
  - On peut lire/utiliser le contenu d'un JAR
- WAR (extension `.war`) :
  - Assemblage de JAR pour une application Web
  - Utilisé pour un déploiement sur un serveur d'application

Maven

Apache Maven™

Gradle : alternative

- Outil de construction qui permet, automatiquement, de :
  - Compiler
  - Gérer les dépendances
  - Exécuter les tests
  - Créer des JAR/WAR
  - D'avoir une structuration uniforme
- Cycle de vie (buts) :
  - `compile`
  - `test`
  - `package`
  - `install`
  - `Deploy`
  - D'autre buts hors cycle de vie : `clean`, `verify`, etc.



Convention plutôt que configuration

Tous les buts en amont d'un but doivent être exécutés sauf s'ils ont déjà été exécutés avec succès et qu'aucun changement n'a été fait dans le projet depuis

Maven

- Référentiels (entrepôts) :
  - Local : `~/.m2/repository`
  - Central : https://repo.maven.apache.org/maven2
  - Privé (optionnel) :
- Commande : `mvn`

```
mvn compile # Compilation.
mvn test # Exécute les tests unitaires.
mvn package # Génère un JAR/WAR.
mvn clean # Supprime le répertoire target.
mvn install # Installe la construction (build) dans le dépôt local.
mvn verify # Génère le rapport de couverture.
```

- Project Object Model : `pom.xml`
  - Métadonnées `groupId`, `artifactId`, `version`
  - Dépendances (librairies externes) : JUnit, JaCoCo, Log4j, etc.
    - Annuaire de dépendances : https://mvnrepository.com
  - Extensions (*plugin*) de tâches spécifiques de construction

Maven est un gestionnaire de paquets

Pour des bibliothèques internes ou afin de cacher le référentiel central pour des raisons de performance ou de sécurité

Peut hériter d'un POM parent

Maven, exemple de 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>com.exemple</groupId>
  <artifactId>mon-projet</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <jacoco.version>0.8.11</jacoco.version>
  </properties>
```

Maven, exemple de POM

```
<dependencies>
    <!-- JUnit Jupiter (JUnit 5) -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.10.2</version>
      <scope>test</scope>
    </dependency>
    <!-- Log4j -->
    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
      <version>2.22.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
      <version>2.22.1</version>
    </dependency>
</dependencies>
```

Maven, exemple de POM

```
<build>
  <plugins>
    <!-- Compiler -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.11.0</version>
    </plugin>

    <!-- Surefire for running JUnit tests -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.2.5</version>
    </plugin>
  </plugins>
</build>

<!-- SonarQube: invocation via CLI or Maven command (not plugin here) -->
</project>
```

JaCoCo

Extension *open source* Maven

JACOCO
Java Code Coverage

- Java Code Coverage
- Mesure la couverture de code des tests unitaires en Java :
  - Évaluer la qualité des tests
  - Identifier le code source non testé
  - Cibler les zones à risques de l'application
  - Générer des rapports en HTML :
    - `target/site/jacoco/index.html`

JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.core		97%		90%	166	1,600	136	3,781	19	769	2	155
org.jacoco.examples		58%		64%	24	53	97	193	19	38	6	12
org.jacoco.agent.rt		75%		83%	32	130	75	344	21	80	7	22
jacoco-maven-plugin		90%		82%	35	193	49	465	8	116	1	23
org.jacoco.cli		97%		100%	4	109	10	275	4	74	0	20
org.jacoco.report		99%		99%	4	572	2	1,345	1	371	0	64
org.jacoco.ant		98%		99%	4	162	8	428	3	110	0	19
org.jacoco.agent		86%		75%	2	10	3	27	0	6	0	1
Total	1,445 of 29,596	95%	206 of 2,466	91%	271	2,829	380	6,858	75	1,564	16	316

JaCoCo, exemple de POM

```
…
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>${jacoco.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>report</id>
      <phase>verify</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin> …
```

À savoir

- Interface `Cloneable`, permet de disposer de la méthode :

```
protected Object clone() { … }
```

- enum :

```
public enum Animal {KANGAROO, TIGER, DOG, SNAKE, CAT, … };
```

- instanceOf :

A utiliser avec parcimonie

```
if (myObj instanceOf MyClass) {
    myObj2 = (MyClass) myObj; // Downcasting.
}
```

Bonnes pratiques

- Traitez toutes les exceptions susceptibles d'être lancées
- Faites attention à ne pas créer de *deadlock*
- Attention à l'héritage d'un générique

Aller plus loin

- Type `record`
- Mot clef `volatile`
- Métaprogrammation par annotation
- Synchronisation de haut niveau (API de concurrence)
- API de management
- Gestion de flux standards : classe `Scanner`
- Site Web avec JHipster

Liens

- Documents électroniques :
  - http://nicolas.baudru.perso.luminy.univ-amu.fr/#PO
- Documents classiques :
  - Livres :
    - Claude Delannoy. *Programmer en Java 2ème édition.*
  - Cours :
    - Francis Jambon. *Programmation orientée application au langage Java.*

Crédits

Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

Relecteurs

Cours en ligne sur : www.mickael-martin-nevot.com