ITR1 Architectures informatiques et réseaux

La machine virtuelle Java

Christophe Garion

ISAE-SUPAERO/DISC

License CC BY-NC-SA 3.0

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license (CC BY-NC-SA 3.0)

You are free to Share (copy, distribute and transmite) and to Remix (adapt) this work under the following conditions :

Attribution – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial – You may not use this work for commercial purposes.

Share Alike – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

See `http://creativecommons.org/licenses/by-nc-sa/3.0/`.

Quote. . .

If Java had true garbage collection, most programs would delete themselves upon execution.

Robert Sewell

Plan

1. **La machine virtuelle Java (JVM)**
2. Le fichier `.class`
3. Organisation mémoire de la JVM
4. Le compilateur JIT
5. Le ramasse-miettes
6. Le chargement dynamique des classes

Une machine virtuelle

La JVM est une machine virtuelle :

- elle fournit une abstraction de la machine sur laquelle on veut faire tourner un programme ;
- c'est une **spécification** ;
- elle permet (en partie) la portabilité des applications Java ;
- elle interprète les programmes Java sous forme de **bytecode** (code intermédiaire).

Une machine virtuelle

Lindholm, T. et F. Yellin (1999).
The Java Virtual Machine.
2e éd.
Addison-Wesley Professional.
https://docs.oracle.com/javase/specs/jvms/se8/html/index.html.

Gosling, J. et al. (2005).
The Java Language Specification.
3e éd.
Java Series.
Prentice Hall.
https://docs.oracle.com/javase/specs/jls/se8/html/index.html.

Une machine virtuelle

Il existe plusieurs implantations de la spécification :

- HotSpot, OpenJDK, Apache Harmony, Jikes RVM
- Maxine, leJos, NanoVM

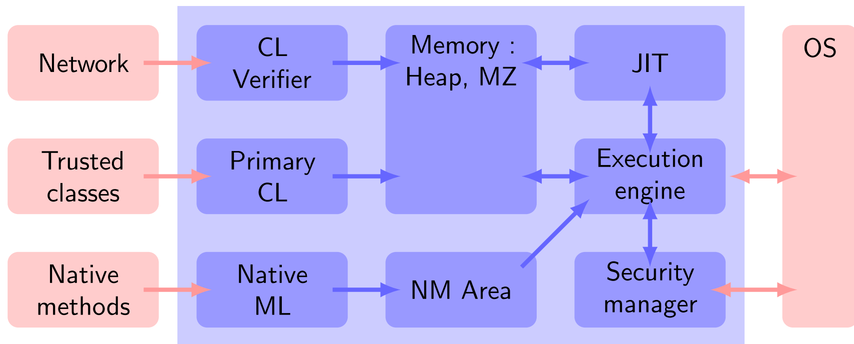
D'autres langages que Java peuvent être compilés vers du *bytecode* pour la JVM :

- Clojure, Groovy, Scala, ...
- JavaScript (Rhino), Python (Jython), Ruby (JRuby), ...

Il existe deux versions de la JVM :

- version client (option `-client`), qui réduit le temps de démarrage et l'occupation mémoire
- version serveur (option `-server`) qui maximise la rapidité d'exécution du programme

Architecture de la JVM



Une machine à pile ?

La JVM est une **machine à pile** : elle fait abstraction du nombre de registres du matériel qui l'accueille. Elle a un jeu d'instructions très simple.

Considérons l'instruction suivante :

```
i = j + k;
```

Voici le code généré sur une machine à pile :

```
push j     // put j on the stack
push k     // put k on the stack
add        // add the two values, remove them
           // put the result on stack
store i    // take the top of stack and store it in i
```

Inconvénients : plus d'accès mémoire
Avantages : pas de décodage et de recherche des opérandes dans les registres (tout est sur la pile !)

Quelques options non standard de la JVM

- `-Xint` : mode interprété pur
- `-Xbatch` : la compilation passe en premier plan
- `-Xfuture` : verification stricte (pas le cas en 1.1)
- `-Xnoclassgc` : pas de ramasse-miettes
- `-Xincgc` : ramasse-miettes incrémental (moins de latence, plus de CPU utilisé)
- `-Xloggc` : log des événements du GC (voir aussi `-verbose:gc`)
- `-Xmsn` : taille initiale du tas (par défaut 2Mo), `n` étant la taille
- `-Xmxn` : taille maximale du tas (par défaut 64Mo), `n` étant la taille
- `-Xprof` : *profiling* du programme sur la sortie standard
- `-Xrunhprof` : *profiling* du CPU, de la pile ou des moniteurs (voir `java -Xrunhprof:help`)

Options avancées de la JVM

Java HotSpot VM Options. http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html.

Java Tuning White Paper. http://www.oracle.com/technetwork/java/tuning-139912.html.

- `-XX:-UseParallelGC` : utilisation du *garbage collector* distribué
- `-XX:-DisableExplicitGC` : plus d'appels à `System.gc()`
- `-XX:+AggressiveOpts` : optimisation agressive (par défaut à partir du SE 6.0)
- `-XX:+UseStringCache` : cache pour les chaînes de caractères souvent utilisées
- ...

Outils à disposition

Certaines options sont gérables directement ou via les JMX (*Java Management Extensions*).

Chung, M.
Monitoring and Managing Java SE 6 Platform Applications.

http://www.oracle.com/technetwork/articles/javase/monitoring-141801.html.

Java SE monitoring and management guide.
https://docs.oracle.com/javase/8/docs/technotes/guides/management/toc.html.

Outils à disposition

Quelques outils (en ligne de commande) :

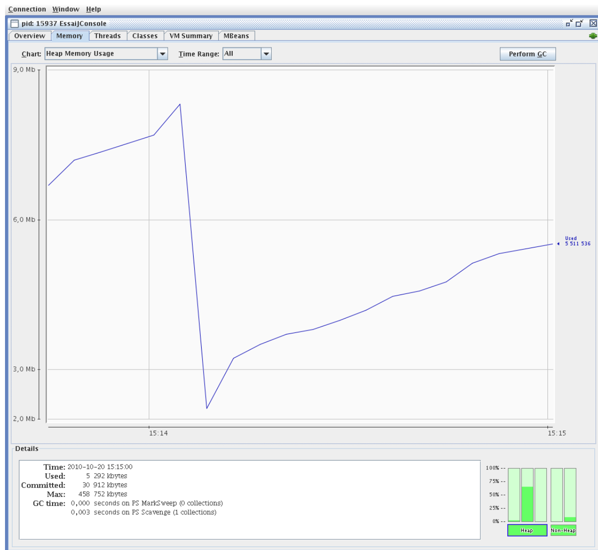
- `jconsole` : outil générique de surveillance d'une application
- `jhat` : analyse du tas
- `jstat` : statistiques diverses
- `jstack` : trace de la pile
- `JVisualVM` : outil générique (seulement dans le SDK d'Oracle, cf. `http://docs.oracle.com/javase/6/docs/technotes/guides/visualvm/index.html`)

Un exemple pour jconsole et consorts

EssaiJConsole.java

```
public class EssaiJConsole {

    /**
     * Describe <code>main</code> method here.
     *
     * @param args a <code>String</code> value
     */
    public static final void main(final String[] args) throws Exception {
        while (true) {
            new Object();
            System.out.println("New object created!");
            Thread.sleep(2000);
        }
    }
}
```

Sortie jconsole



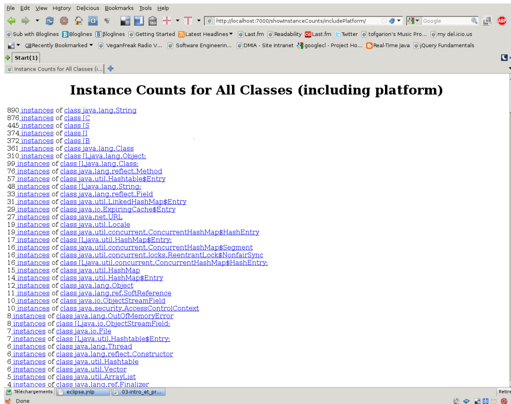
Utilisation de jmap

```
jmap -histo:live <pid>
```

```
 num     #instances         #bytes  class name
----------------------------------------------
   1:          4661         460224  <constMethodKlass>
   2:          4661         375232  <methodKlass>
   3:          7236         264944  <symbolKlass>
   4:           305         176128  <constantPoolKlass>
   5:           305         124288  <instanceKlassKlass>
   6:           278         123024  <constantPoolCacheKlass>
   7:           876          75072  [C
   8:           372          72224  [B
   9:           374          35904  java.lang.Class
  10:           482          29576  [[I
  11:           445          28016  [S
  12:           890          21360  java.lang.String
  13:           374          19168  [I
  14:            40          13120  <objArrayKlassKlass>
  15:           310          12792  [Ljava.lang.Object;
  16:            76           6080  java.lang.reflect.Method
  17:             8           2624  <typeArrayKlassKlass>
  18:            33           2376  java.lang.reflect.Field
```

Utilisation de jhat

```
jmap -dump:live,file=heap.dump.out <pid>
jhat heap.dump.out
```



Plan

1. La machine virtuelle Java (JVM)
2. **Le fichier `.class`**
3. Organisation mémoire de la JVM
4. Le compilateur JIT
5. Le ramasse-miettes
6. Le chargement dynamique des classes

Qu'y a-t-il dans un fichier `.class` ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

Qu'y a-t-il dans un fichier **`.class`** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

4 premiers octets : magic number
4 octets suivants : numéros de version

Qu'y a-t-il dans un fichier **`.class`** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

Zone des « constantes » : tableau qui contient toutes les informations symboliques typées.

- *CONSTANT_Class*
- *CONSTANT_Fieldref*
- *CONSTANT_Methodref*
- *CONSTANT_...* pour les types primitifs
- *CONSTANT_NameAndType*
- *CONSTANT_Utf8*

Dans la suite du fichier, tous les symboles sont référencés par des index sur ce tableau.

Qu'y a-t-il dans un fichier **`.class`** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

access flags : classe, interface, publique, abstraite, finale ?

Qu'y a-t-il dans un fichier **`.class`** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

this_class et **super_class** : index dans la zone des constantes

Qu'y a-t-il dans un fichier **`.class`** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

- `0xCAFEBABE`
- `constants`
- `access flags`
- **`class`**
- `superclass`
- `interfaces`
- `fields`
- `methods`
- `properties`

Partie **interfaces** : le nombre d'interfaces réalisées et les index vers la table des constantes pour leurs noms.

Qu'y a-t-il dans un fichier `.class` ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

Attributs :

- nombre
- nom (zone des constantes)
- valeur initiale éventuelle
- ...

Qu'y a-t-il dans un fichier **.class** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **.class** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

0xCAFEBABE
constants
access flags
class
superclass
interfaces
fields
methods
properties

Méthodes : nombre, nom (zone des constantes), signature, mais également

- réservation dans la pile pour les variables locales
- exceptions
- *bytecode* à exécuter
- numéros de lignes

Qu'y a-t-il dans un fichier **`.class`** ?

Les sources Java sont compilés en *bytecode* contenu dans des fichiers **`.class`** clairement définis.

Dans ces fichiers, on trouve un certain nombre d'informations :

`0xCAFEBABE`
`constants`
`access flags`
`class`
`superclass`
`interfaces`
`fields`
`methods`
`properties`

Propriétés : informations générales (fichier source etc.)

Fichier **.class** : un exemple

Personne.java

```
public class Personne {

    private String nom;
    private int age;

    public Personne(String nom, int age) {
        this.nom = nom.trim();
        this.age=age;
    }

    public String getNom() {
        return this.nom;
    }

    public int getAge() {
        return this.age;
    }
}
```

Fichier **.class** : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

Fichier .class : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

0000 0031

➥ major version 49

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a   .......1........
0016 0017 0900 0500 1809 0005 0019 0700   ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61   .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01   va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69   ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67   t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f   /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254   de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014   able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472   ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003   ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65   ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661   ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007   ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e   ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f   ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61   bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100   va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200   .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000   ................
0300 0100 0b00 0c00 0100 0d00 0000 3600   ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002   .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00   ...*............
0000 1200 0400 0000 0600 0400 0700 0c00   ................
0800 1100 0900 0100 0f00 1000 0100 0d00   ................
0000 1d00 0100 0100 0000 052a b400 03b0   ...........*....
0000 0001 000e 0000 0006 0001 0000 000c   ................
0001 0011 0012 0001 000d 0000 001d 0001   ................
0001 0000 0005 2ab4 0004 ac00 0000 0100   ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300   ................
```

001f

➥ 31 constantes utilisées

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

09 0005 0018

- ➥ fieldref
- ➥ class #5
- ➥ name and type #24

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

07 001a

➥ class

➥ name #26

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

01 0008 5065...

➥ UTF-8 : `Personne`

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

09 0005 0018

- ➥ field
- ➥ class #5
- ➥ name and type #24

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a  .......1........
0016 0017 0900 0500 1809 0005 0019 0700  ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61  .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01  va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69  ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67  t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f  /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254  de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014  able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472  ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003  ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65  ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661  ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007  ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e  ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f  ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61  bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100  va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200  .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000  ................
0300 0100 0b00 0c00 0100 0d00 0000 3600  ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002  .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00  ...*............
0000 1200 0400 0000 0600 0400 0700 0c00  ................
0800 1100 0900 0100 0f00 1000 0100 0d00  ................
0000 1d00 0100 0100 0000 052a b400 03b0  ...........*....
0000 0001 000e 0000 0006 0001 0000 000c  ................
0001 0011 0012 0001 000d 0000 001d 0001  ................
0001 0000 0005 2ab4 0004 ac00 0000 0100  ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300  ................
```

0c 0007 0008

- ➥ name and type
- ➥ name #7
- ➥ type name #8

Fichier `.class` : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a   .......1........
0016 0017 0900 0500 1809 0005 0019 0700   ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61   .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01   va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69   ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67   t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f   /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254   de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014   able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472   ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003   ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65   ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661   ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007   ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e   ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f   ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61   bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100   va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200   .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000   ................
0300 0100 0b00 0c00 0100 0d00 0000 3600   ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002   .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00   ...*............
0000 1200 0400 0000 0600 0400 0700 0c00   ................
0800 1100 0900 0100 0f00 1000 0100 0d00   ................
0000 1d00 0100 0100 0000 052a b400 03b0   ...........*....
0000 0001 000e 0000 0006 0001 0000 000c   ................
0001 0011 0012 0001 000d 0000 001d 0001   ................
0001 0000 0005 2ab4 0004 ac00 0000 0100   ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300   ................
```

0100 03 6e6f6d

➥ UTF-8 : `nom`

Fichier .class : format hexadécimal

```
cafe babe 0000 0031 001f 0a00 0600 150a   .......1........
0016 0017 0900 0500 1809 0005 0019 0700   ................
1a07 001b 0100 036e 6f6d 0100 124c 6a61   .......nom...Lja
7661 2f6c 616e 672f 5374 7269 6e67 3b01   va/lang/String;.
0003 6167 6501 0001 4901 0006 3c69 6e69   ..age...I...<ini
743e 0100 1628 4c6a 6176 612f 6c61 6e67   t>...(Ljava/lang
2f53 7472 696e 673b 4929 5601 0004 436f   /String;I)V...Co
6465 0100 0f4c 696e 654e 756d 6265 7254   de...LineNumberT
6162 6c65 0100 0667 6574 4e6f 6d01 0014   able...getNom...
2829 4c6a 6176 612f 6c61 6e67 2f53 7472   ()Ljava/lang/Str
696e 673b 0100 0667 6574 4167 6501 0003   ing;...getAge...
2829 4901 000a 536f 7572 6365 4669 6c65   ()I...SourceFile
0100 0d50 6572 736f 6e6e 652e 6a61 7661   ...Personne.java
0c00 0b00 1c07 001d 0c00 1e00 100c 0007   ................
0008 0c00 0900 0a01 0008 5065 7273 6f6e   ..........Person
6e65 0100 106a 6176 612f 6c61 6e67 2f4f   ne...java/lang/O
626a 6563 7401 0003 2829 5601 0010 6a61   bject...()V...ja
7661 2f6c 616e 672f 5374 7269 6e67 0100   va/lang/String..
0474 7269 6d00 2100 0500 0600 0000 0200   .trim.!.........
0200 0700 0800 0000 0200 0900 0a00 0000   ................
0300 0100 0b00 0c00 0100 0d00 0000 3600   ..............6.
0200 0300 0000 122a b700 012a 2bb6 0002   .......*...*+...
b500 032a 1cb5 0004 b100 0000 0100 0e00   ...*............
0000 1200 0400 0000 0600 0400 0700 0c00   ................
0800 1100 0900 0100 0f00 1000 0100 0d00   ................
0000 1d00 0100 0100 0000 052a b400 03b0   ...........*....
0000 0001 000e 0000 0006 0001 0000 000c   ................
0001 0011 0012 0001 000d 0000 001d 0001   ................
0001 0000 0005 2ab4 0004 ac00 0000 0100   ......*.........
0e00 0000 0600 0100 0000 1000 0100 1300   ................
```

Méthode getNom

0001	**public**
000f	getNom
0010	retour et args
0001	attributes
000d	Code
..1d	longueur (29)
0001	max. *stack*
0001	max. var.
..05	long. code
2a	aload_0
b4	getfield
0003	#3
b0	areturn
...	ligne

Fichier .class : désassemblage

Désassemblage avec `javap -verbose -c Personne` :

```
Compiled from "Personne.java"
public class Personne extends java.lang.Object
  SourceFile: "Personne.java"
  minor version: 0
  major version: 49
  Constant pool:
const #1 = Method       #6.#21; //  java/lang/Object."<init>":()V
const #2 = Method       #22.#23;        //  java/lang/String.trim:()Ljava/lang/String;
const #3 = Field        #5.#24; //  Personne.nom:Ljava/lang/String;
const #4 = Field        #5.#25; //  Personne.age:I
const #5 = class        #26;    //  Personne
const #6 = class        #27;    //  java/lang/Object
const #7 = Asciz        nom;
const #8 = Asciz        Ljava/lang/String;;
const #9 = Asciz        age;

...

const #20 = Asciz       Personne.java;
const #21 = NameAndType #11:#28;//  "<init>":()V
const #22 = class       #29;    //  java/lang/String
const #23 = NameAndType #30:#16;//  trim:()Ljava/lang/String;
const #24 = NameAndType #7:#8;//  nom:Ljava/lang/String;
const #25 = NameAndType #9:#10;//  age:I
const #26 = Asciz       Personne;
const #27 = Asciz       java/lang/Object;
const #28 = Asciz       ()V;
const #29 = Asciz       java/lang/String;
...
```

Fichier .class : désassemblage

```
{
public Personne(java.lang.String, int);
  Code:
   Stack=2, Locals=3, Args_size=3
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   aload_0
   5:   aload_1
   6:   invokevirtual   #2; //Method java/lang/String.trim:()Ljava/lang/String;
   9:   putfield        #3; //Field nom:Ljava/lang/String;
   12:  aload_0
   13:  iload_2
   14:  putfield        #4; //Field age:I
   17:  return
  LineNumberTable:
   line 6: 0
   line 7: 4
   line 8: 12
   line 9: 17
```

Fichier .class : désassemblage

```
public java.lang.String getNom();
  Code:
   Stack=1, Locals=1, Args_size=1
   0:   aload_0
   1:   getfield        #3; //Field nom:Ljava/lang/String;
   4:   areturn
  LineNumberTable:
   line 12: 0

public int getAge();
  Code:
   Stack=1, Locals=1, Args_size=1
   0:   aload_0
   1:   getfield        #4; //Field age:I
   4:   ireturn
  LineNumberTable:
   line 16: 0

}
```

Désassemblage : mais que fait le compilateur ?

TryJavap.java

```
import java.util.ArrayList;

class TryJavap {
    public static void main(String[] args) {
        ArrayList<Object> l = new ArrayList<Object>();

        for (Object o: l) {
            System.out.println(o);
        }
    }
}
```

Désassemblage : mais que fait le compilateur ?

```
public static void main(java.lang.String[]);
  descriptor: ([Ljava/lang/String;)V
  flags: ACC_PUBLIC, ACC_STATIC
  Code:
    stack=2, locals=4, args_size=1
       0: new           #2                  // class java/util/ArrayList
       3: dup
       4: invokespecial #3                  // Method java/util/ArrayList."<init>":()V
       7: astore_1
       8: aload_1
       9: invokevirtual #4                  // Method java/util/ArrayList.iterator:()L
      12: astore_2
      13: aload_2
      14: invokeinterface #5,  1            // InterfaceMethod java/util/Iterator.hasN
      19: ifeq          39
      22: aload_2
      23: invokeinterface #6,  1            // InterfaceMethod java/util/Iterator.next
      28: astore_3
      29: getstatic     #7                  // Field java/lang/System.out:Ljava/io/Pri
      32: aload_3
      33: invokevirtual #8                  // Method java/io/PrintStream.println:(Lja
      36: goto          13
      39: return
```

Méthode getNom : que se passe-t-il ?

Code mnémoniques :

```
0:   aload_0
1:   getfield        #3; //Field nom:Ljava/lang/String;
4:   areturn
```

Pile getNom

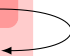
...

Heap

:Personne

age : 31

nom : ref



:String

Christophe

Méthode getNom : que se passe-t-il ?

Code mnémoniques :

```
0:    aload_0
1:    getfield        #3; //Field nom:Ljava/lang/String;
4:    areturn
```

aload_0 [. . . ⇒ . . . , objectref]

aload_0 place la référence vers l'objet index #0 sur la pile

➥ il s'agit de **this**

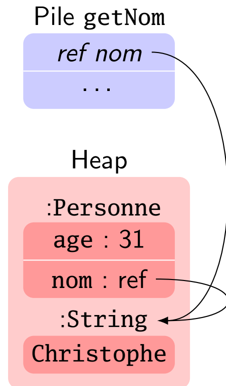
Méthode getNom : que se passe-t-il ?

Code mnémoniques :

```
0:   aload_0
1:   getfield        #3; //Field nom:Ljava/lang/String;
4:   areturn
```

getfield [..., objectref ⇒ ..., value]

getfield #3 prend la valeur sur la pile (c'est une référence) et utilise l'index #3 de la table des constantes pour obtenir un attribut.
On résoud ensuite « l'attribut », on prend sa valeur dans la référence obtenue sur la pile et on place cette valeur sur la pile.



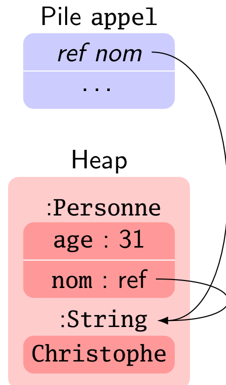
Méthode `getNom` : que se passe-t-il ?

Code mnémoniques :

```
0:   aload_0
1:   getfield         #3; //Field nom:Ljava/lang/String;
4:   areturn
```

areturn [..., objectref ⇒ ...]

areturn prend la référence sur la pile et la met sur la pile de la méthode appelante.
Le reste de la pile est effacé.
Le contrôle est redonné à la méthode appelante.



Plan

1. La machine virtuelle Java (JVM)
2. Le fichier `.class`
3. **Organisation mémoire de la JVM**
4. Le compilateur JIT
5. Le ramasse-miettes
6. Le chargement dynamique des classes

Organisation mémoire : grands principes

4 grandes zones :

- la **zone des méthodes** qui contient le code à exécuter
- le **tas** qui contient les objets alloués par **`new`**
- la **pile** qui contient les contextes (**frames**) d'invocation des méthodes. Elle est gérée par les registres `optop`, `frame` et `vars`
- les **registres** : les trois registres de piles et le compteur ordinal

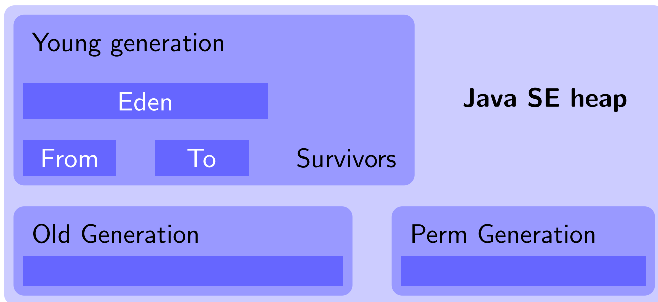
Attention :

	JVM	thread
Method area	×	
Heap	×	
Stack		×
Registries		×

S'il n'y a plus de place, une erreur `OutOfMemoryError` est levée.

Le tas (*heap*) et la zone des méthodes

La zone du tas (*heap*) est l'endroit où vivent les objets Java.



Elle est automatiquement gérée par un processus, le ramasse-miettes (*garbage-collector*).

La zone des méthodes

La zone des méthodes est analogue avec la zone mémoire contenant le code à exécuter dans le cas d'un code compilé.

Elle contient le *bytecode* des méthodes et des constructeurs (`<init>`) des classes.

Le compteur de programme pointe dessus.

Elle contient également la table des constantes permettant l'identification des constantes des classes.

Elle devrait faire partie du tas, mais on peut choisir de ne pas utiliser de GC dessus par exemple.

La zone des méthodes

Jusqu'à Java 7, la zone des méthodes était stockée dans la PermGen et géré comme le tas.

Depuis Java 8, la zone des méthodes est stockée dans un espace mémoire natif, le **Metaspace**.

Masamitsu, Jon (2014).
JEP 122 : Remove the Permanent Generation.
http://openjdk.java.net/jeps/122.

Pile, contexte et registres

La pile Java stocke les paramètres, les résultats intermédiaires, les retours de méthode etc.

Pour chaque méthode exécutée, il existe un contexte d'exécution (*stack frame*) qui est une certaine zone de la pile :

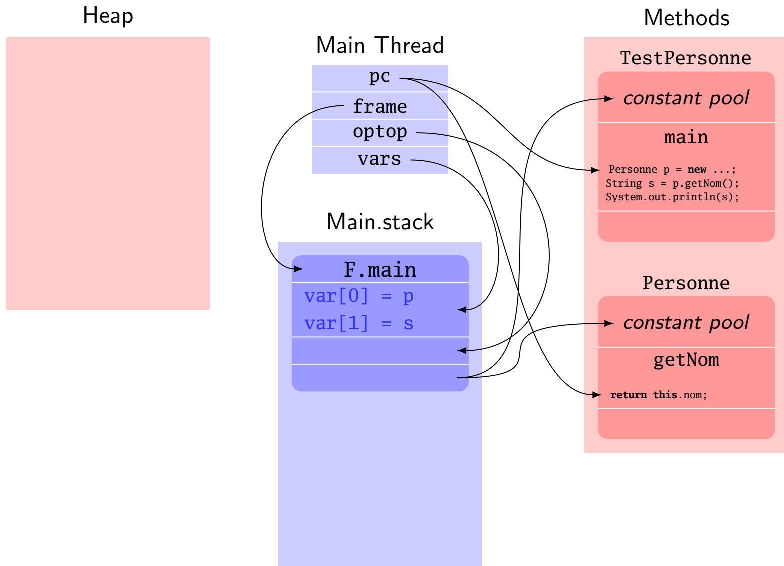
- zone des variables locales (pointée par `vars`) représentée par un tableau. Cette zone contient également **`this`** (index 0) et les paramètres de la méthode.
- zone opérandes (pointée par `optop`) : paramètres des instructions etc. C'est une pile de longueur maximale déterminée à la compilation. Elle contient des valeurs des types défini dans Java.
- zone d'environnement d'exécution (pointée par `frame`) pour conserver des informations sur les manipulations de la pile.
- la pile contient également une référence vers la zone *constant pool* de la classe de la méthode.

Organisation mémoire de la JVM : résumé

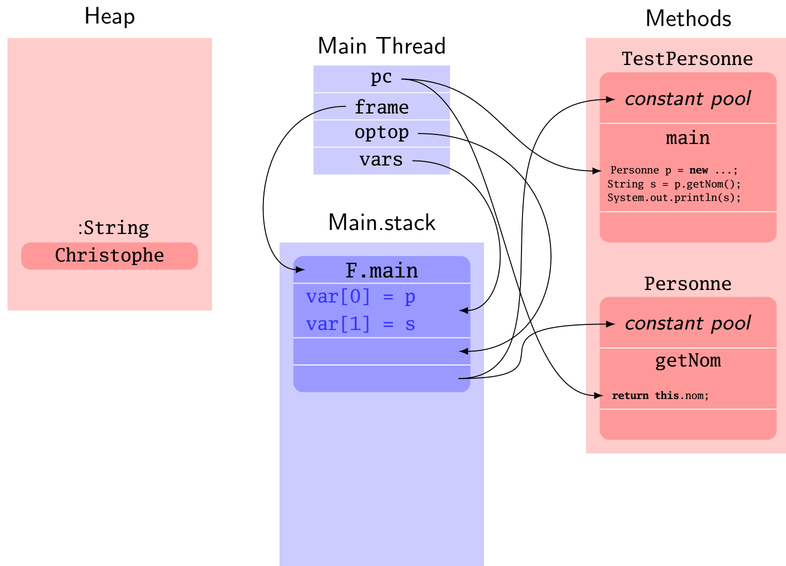
TestPersonne.java

```
public class TestPersonne {

    public static final void main(final String[] args) {
        Personne p = new Personne("Christophe", 39);
        String s = p.getNom();
        System.out.println(s);
    }
}
```

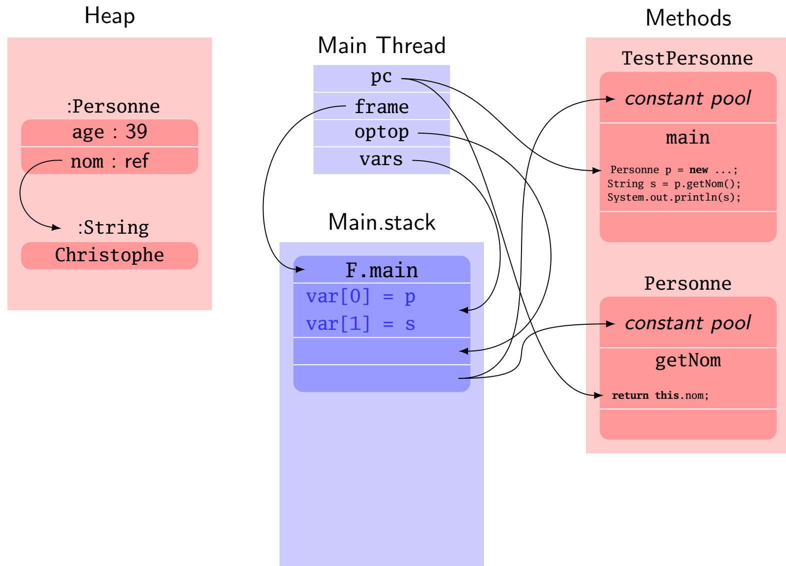
Organisation mémoire de la JVM : résumé



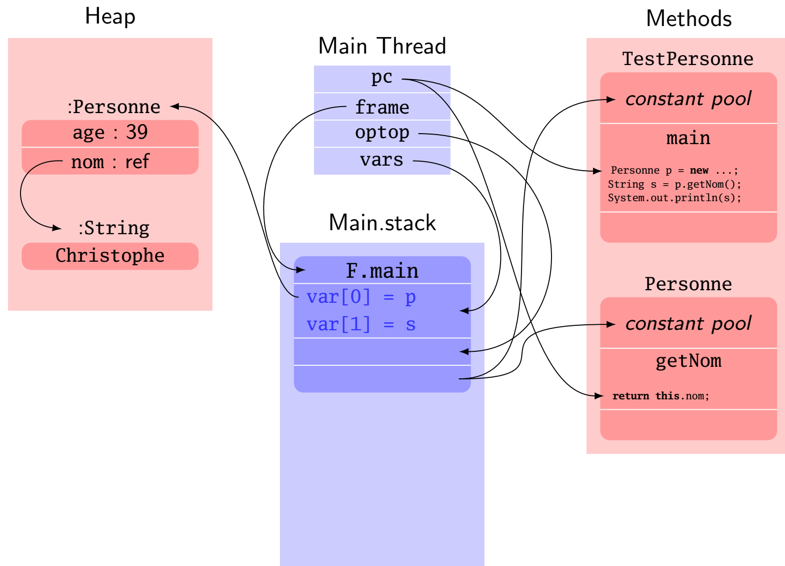
Organisation mémoire de la JVM : résumé



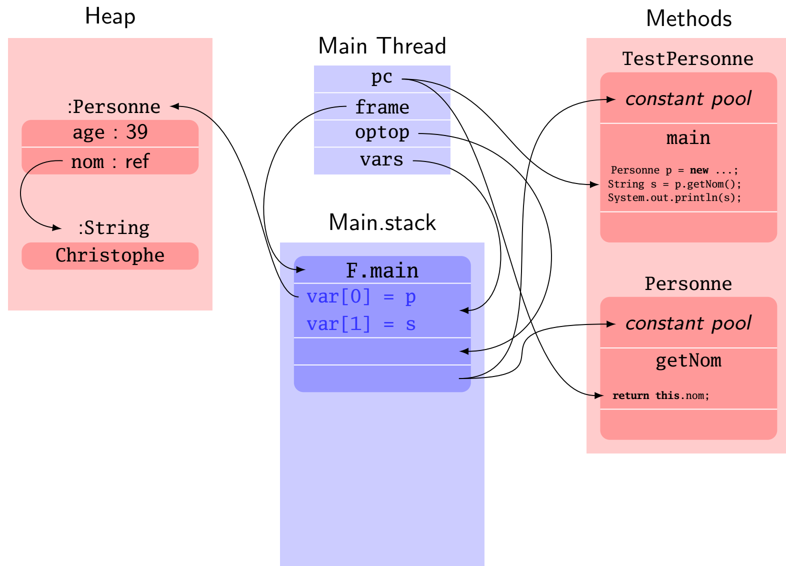
Organisation mémoire de la JVM : résumé



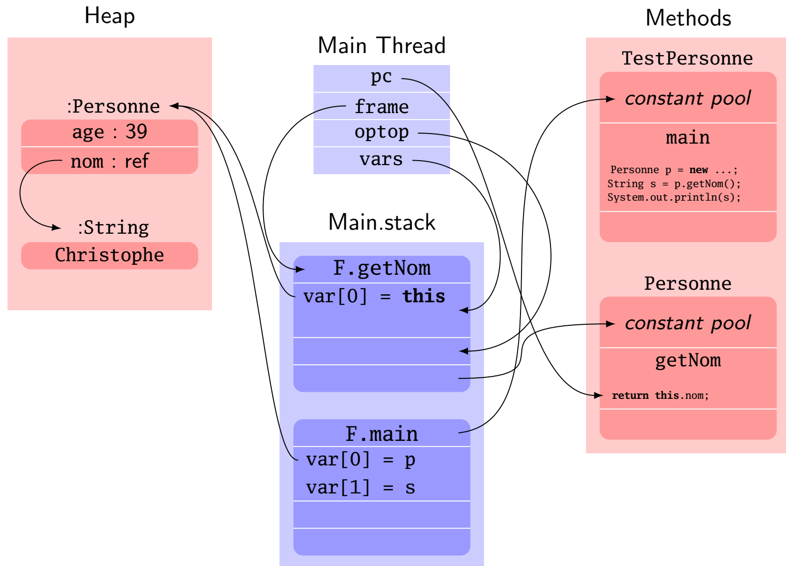
Organisation mémoire de la JVM : résumé



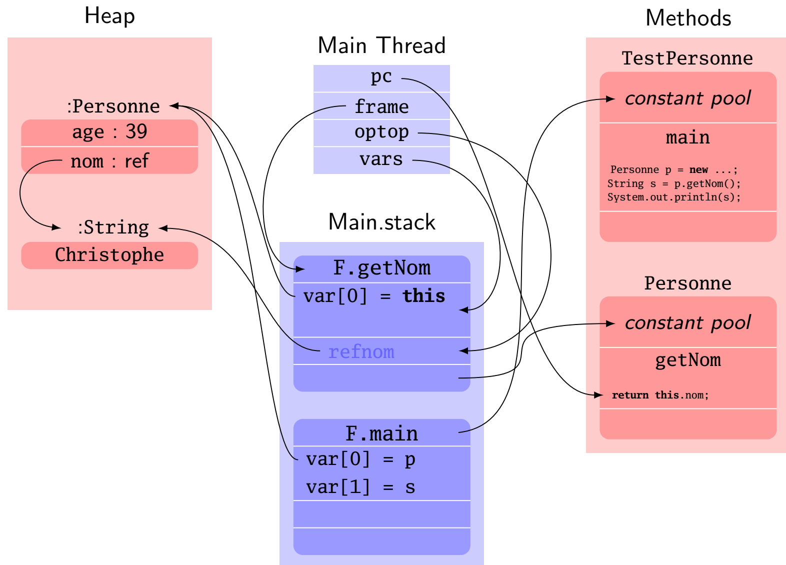
Organisation mémoire de la JVM : résumé



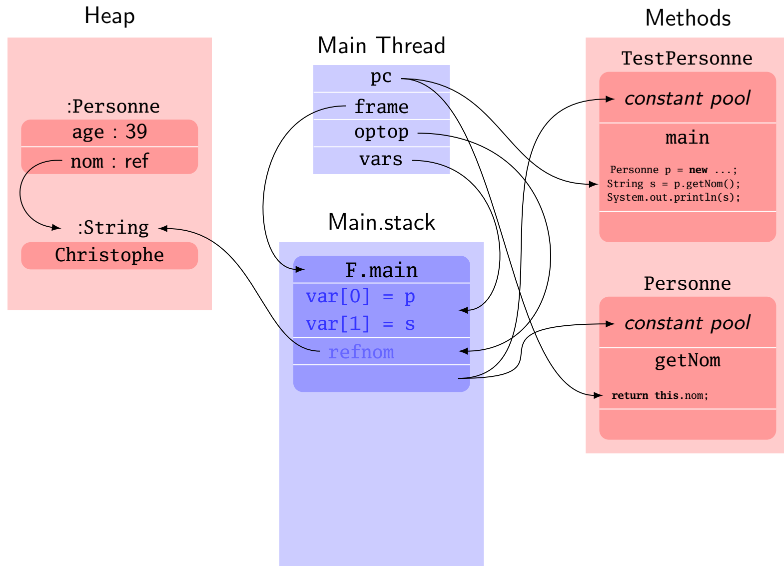
Organisation mémoire de la JVM : résumé



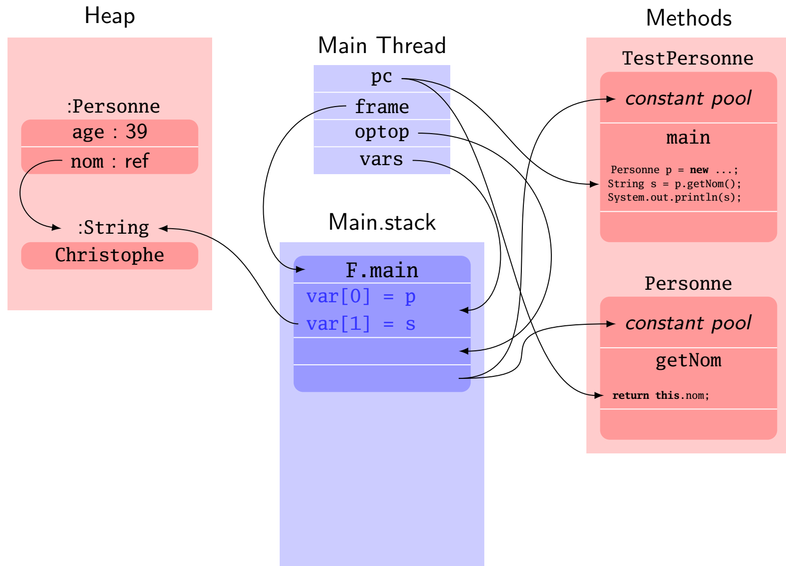
Organisation mémoire de la JVM : résumé



Organisation mémoire de la JVM : résumé



Organisation mémoire de la JVM : résumé



Manque de mémoire sur le tas : exemple

```java
public class MemErr {
    public static void main(String[] args) throws Exception {
        Thread.sleep(5000);

        Cellule aux = null;

        for (int i = 0; i < Integer.MAX_VALUE; i++) {
            aux = new Cellule(aux);
        }
    }
}

class Cellule {
    Cellule suivante;

    Cellule (Cellule suivante_) {
        this.suivante  = suivante_;
    }
}
```

Manque de mémoire sur le tas : exemple

```
% java MemErr
```

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at MemErr.main(MemErr.java:8)
```

Quelles erreurs ?

- `heap space` (ici) : pas de place pour allouer de la mémoire pour un objet
- `PermGen space` : pas de place pour charger une classe ou stocker une chaîne de caractères interne
- *native memory error* : problème d'allocation sur la pile native

Plan

1. La machine virtuelle Java (JVM)
2. Le fichier .class
3. Organisation mémoire de la JVM
4. **Le compilateur JIT**
5. Le ramasse-miettes
6. Le chargement dynamique des classes

Le compilateur JIT

L'interprétation est très pénalisante en terme de performances (par exemple pour les boucles) : utilisation d'un **compilateur Just in Time**.

Est-ce que le compilateur JIT compile chaque méthode ?

- temps de compilation préjudiciable pour l'utilisateur
- même s'il pouvait optimiser complétement le code, ça ne marcherait pas pour Java du fait de son caractère dynamique :
    - vérification dynamique de types fréquentes (*cast*, tableaux) pour la sûreté
    - objets alloués sur le tas (pas toujours le cas pour C++)
    - méthodes principalement virtuelles, donc *inlining* difficile
    - chargement dynamique des classes, donc optimisation globale difficile pour le compilateur

La JVM HotSpot

La JVM HotSpot :

- démarre un interpréteur et lance le programme
- analyse le code pendant qu'il tourne et détecte les points chauds qui sont exécutés souvent (ils sont quand même interprétés !)
- données collectées pendant l'exécution (temps, appel de méthodes virtuelles etc.)
- *inline* le code : moins d'appel de méthodes et optimisation plus efficace (code plus gros)
- déoptimisation/réoptimisation dynamique (à cause du chargement dynamique)

La JVM HotSpot

Quelques optimisations du compilateur :

- *inlining* comme vu précédemment
- détection de type efficace
- élimination de vérification de dépassement de tableau
- boucles « déroulées »
- *profiling* avant compilation en natif

Un exemple d'inlining sur la JVM

FastJavaFast.java

```java
public class FastJavaFast {
  public void test() { for (int i = 0; i < 1000 * 1000 * 1000; i++)
      test2();  }

  public void test2() { for (int i = 0; i < 1000 * 1000 * 1000; i++)
      test3(); }

  public void test3() { for (int i = 0; i < 1000 * 1000 * 1000; i++)
      foo(); }

  public void foo() { }

  public static void main(String[] args) {
    long time = System.currentTimeMillis();
    FastJavaFast fast = new FastJavaFast();
    fast.test();
    time = System.currentTimeMillis() - time;
    System.out.println("time = " + time);
  }
}
```

Exécution de FastJavaFast

```
[tof@suntof]~/Cours/IN329/Transparents/JVM/src/jit % java FastJavaFast
  1%      FastJavaFast::test3 @ 2 (19 bytes)
  1       FastJavaFast::test3 (19 bytes)
  2%      FastJavaFast::test2 @ 2 (19 bytes)
  2       FastJavaFast::test2 (19 bytes)
  3%      FastJavaFast::test @ 2 (19 bytes)
time = 11
```

. . . alors qu'avec `-X:int`. . .

% signifie que le compilateur utilise une optimisation OSR (*On Stack Replacement*).

Production de code assembleur

Avec l'option `-XX:PrintOptoAssembly` (nécessite une JVM *debug*) :

```
{method}
 - klass: {other class}
 - this oop:          0x8fcd9da0
 - method holder:     'FastJavaFast'

...

000   N89: #    B1 <- BLOCK HEAD IS JUNK   Freq: 1
000     INT3
        NOP     # 3 bytes pad for loops and calls

008   B1: #     B8 B2 <- BLOCK HEAD IS JUNK   Freq: 1
008     # stack bang
        PUSHL  EBP
        SUB    ESP,24   # Create frame
016     MOV    EDI,[ECX]        # int
018     MOV    EBX,[ECX + #4]

...
```

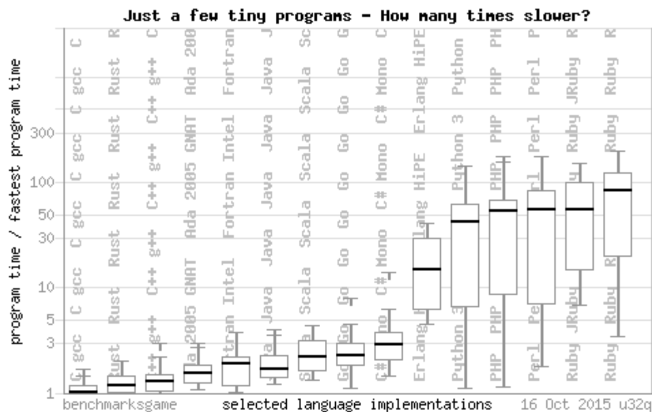
Alors, est-ce que ça marche ?

Java plus rapide que C ?

P.Lewis, J. et U. Neumann (2004).
Performance of Java versus C++.
http://www.idiom.com/~zilla/Computer/javaCbenchmark.html.

Alors, est-ce que ça marche ?

http://benchmarksgame.alioth.debian.org/



Plan

1. La machine virtuelle Java (JVM)
2. Le fichier .class
3. Organisation mémoire de la JVM
4. Le compilateur JIT
5. **Le ramasse-miettes**
6. Le chargement dynamique des classes

Problématique

Le ramasse-miette est un processus chargé de libérer la mémoire allouée dans le tas par l'opérateur **`new`**.

Les objets qui ne sont plus référencés dans le programme sont des candidats potentiels au recyclage.

- permet d'éviter la fragmentation et l'augmentation de la taille du tas ;
- libère le programmeur de la tâche de libérer la mémoire (en particulier quand la libérer) ;
- sécurité ;
- mais charge de calcul supplémentaire.

Ça fonctionne : création de 100000 d'instances d'`Object` et trace avec `-verbose:gc`

```
[GC 512K->122K(1984K), 0.0028650 secs]
```

Comment recycler la mémoire ?

Deux objectifs pour le ramasse-miettes :

- trouver les objets récupérables
- réclamer l'espace du tas utilisé par ces objets

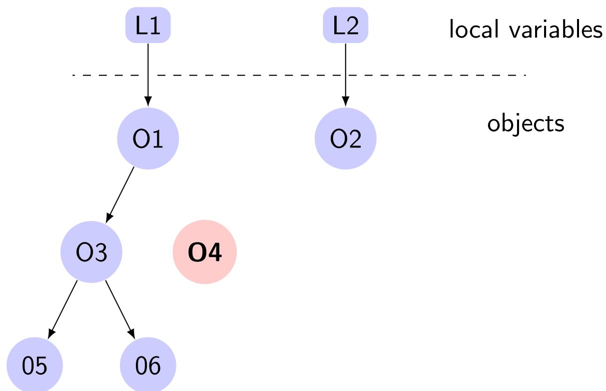
C'est un champ d'étude à part entière !

Quelques possibilités :

- GC parallèle aux threads applicatifs (*concurrent* GC)
- GC parallèle à d'autres activités GC (*parallel* GC)
- séquentiel
- lancé lors d'allocation d'objets
- lancé lorsqu'une limite mémoire est atteinte
- ...

GC par traçage

On détermine les objets atteignables à partir de points d'ancrage.



Pas très efficace, on utilise le scope des variables...

Stop-the-World, Incrémental et Concurrent

Stop-the-world GC :

- on arrête tout lorsque le GC se déclenche
- en particulier pas de création d'objets
- temps de latence de l'application important

GC incrémental :

- on sépare le travail du GC en plusieurs petites parties
- risque : on n'arrive pas à limiter la consommation mémoire

GC concurrent :

- en parallèle de l'application
- idéal pour les systèmes multi-cœurs ou multi-processeurs

Lutter contre la fragmentation

Compactage :

- on place tous les objets dans une extrémité libre du tas
- l'autre extrémité est libre
- modification des références via un degré d'indirection ou pas

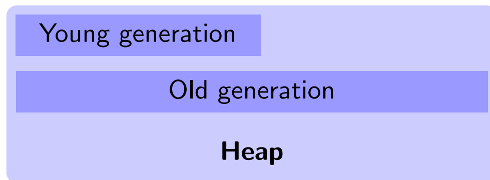
Copie :

- on déplace les objets vers une nouvelle zone
- on les colle les uns aux autres
- on peut les déplacer au fur et à mesure de leur découverte

GC générationnel

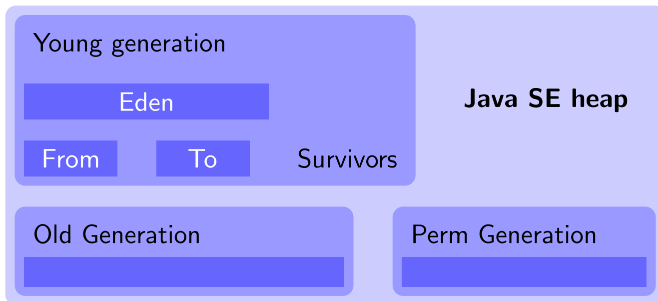
Principe

Les objets à récupérer sont le plus souvent ceux nouvellement créés.



- les objets nouvellement alloués sont dans l'espace Young
- on les promeut dans l'espace Old quand ils survivent au GC
- deux types de GC

Allocation dans le tas de la JVM et GC



- les objets survivant à un *minor collection* passent dans To ;
- on peut les proumouvoir dans Old ;
- *major collection* pour Old qui optimise la fragmentation.

Différents GC pour Java

Serial Collector (`-XX:+UseSerialGC`)

- *mark*, *sweep*, *sliding-compact phase*
- *stop-the-world* même sur du multi-cœurs
- possible sur les deux générations

Parallel Scavenging Collector (`-XX:+UseParallelGC`)

- fonctionne de la même façon que le *serial collector*
- utilise le parallèlisme
- que sur Young

Parallel-Compacting Collector (`-XX:+UseParallelOldGC`)

- fonctionne de la même façon que le *parallel scavenging collector* sur Young
- sur Old, fonctionne comme le *serial collector*, mais découpe en régions

Différents GC pour Java

Concurrent Mark-Sweep Collector (`-XX:+UseConcMarkSweepGC`)

- que sur Old
- évite une trop longue latence sur les collectes
- algorithme
  1. marquage initial (*first-level* objects) [pause]
  2. marquage concurrent [no pause]
  3. nettoyage concurrent [no pause]
  4. remarquage [pause]
  5. balayage concurrent [no pause]
- **fragmentation**
- `-XX:+CMSIncrementalMode`

Différents GC pour Java

L'avenir : Garbage-First (G1)

- tas séparé en *cards* de 512 octets
- on travaille en parallèle sur les *cards*
- bonnes performances

Oracle (2015).
Java Garbage Collection Basics.
http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html.

Peut-on interagir avec le GC depuis Java ?

Comment interagir avec le ramasse-miettes depuis Java ?

La méthode `finalize` d'`Object` :

- **`public void`** `finalize()` **`throws`** `Throwable`
- appelée avant la réclamation de l'espace mémoire de l'objet
- utile pour les ressources ouvertes
- utiliser **`super`**`.finalize`
- résurrection unique d'un objet

Des méthodes « utiles » (*best effort*) :

- `System.gc()`
- `System.runFinalization()`
- `Runtime.gc()`

Une modélisation des références en Java

Peut-on « ramasser » un objet même si l'on a une référence dessus (exemple d'images dans une application avec ascenseurs) ?

On peut utiliser des objets références dont le rôle est de maintenir une référence vers un objet référent. On n'utilise pas alors de poignée directe sur l'objet.

On peut pour cela utiliser la classe abstraite `java.lang.ref.Reference` :

- `public Object get()`
- `public void clear()`
- `public boolean enqueue()`
- `public boolean isEnqueued()`

Force des références et cycle de vie d'un objet

On dispose de trois classes de références : `SoftReference`, `WeakReference`, `PhantomReference`.

Elles correspondent aux différents états d'atteignabilité d'un objet :

- les références normales sont appelées **références fortes**. Un objet est dit **atteignable fortement** s'il existe une chaine de références fortes vers cet objet ;
- un objet « **légérement** » **atteignable** n'est pas atteignable fortement, mais atteignable par une chaine qui contient une réference légère. Il peut être ramassé ;
- un objet **faiblement atteignable** n'est pas atteignable « légérement », mais atteignable par une chaine qui contient une réference faible. Il sera ramassé ;
- un objet est « **finalize reachable** » s'il n'est plus faiblement atteignable, mais il n'a pas été finalisé ;
- un objet est « **phantom reachable** » s'il a été finalisé mais est atteignable par une chaine contenant une référence fantôme. Sa méthode `get` renvoie **`null`** ;
- un objet n'est plus atteignable s'il n'est plus atteint par aucune référence.

Références : un exemple

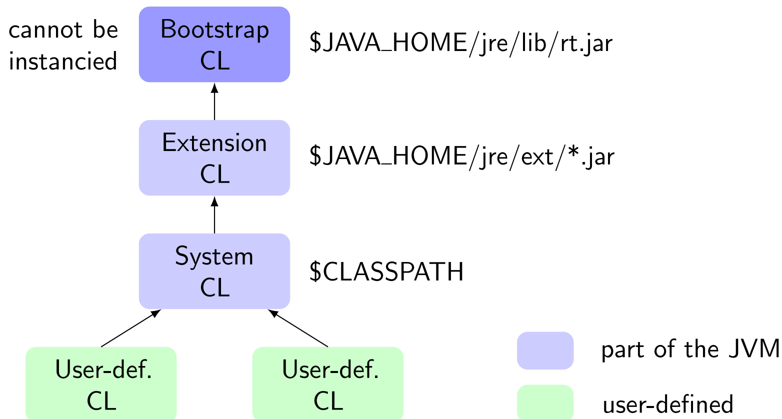
DataHandler.java

```java
import java.lang.ref.*;
import java.io.File;

public class DataHandler {
    private File lastFile;
    private WeakReference lastData;

    byte[] readFile(File file) {
        byte[] data;

        if (file.equals(lastFile)) {
            data = (byte[]) lastData.get();
            if (data != null) {
                return data;
            }
        }

        data = readBytesFromFile(file);
        lastFile = file;
        lastData = new WeakReference(data);
        return data;
    }
}
```

Plan

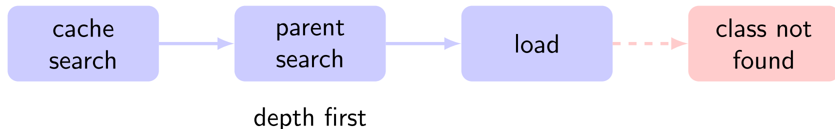
1. La machine virtuelle Java (JVM)
2. Le fichier .class
3. Organisation mémoire de la JVM
4. Le compilateur JIT
5. Le ramasse-miettes
6. **Le chargement dynamique des classes**

Architecture du *class loader*



Architecture du *Class loader*

Fonctionnement lors de la demande de chargement d'une classe :



Peut lever une `ClassNotFoundException`.

Class loader et espace de nommage

Lorsqu'une classe est chargée, la JVM « retient » quel *class loader* a été utilisé.

On utilise le même *class loader* pour les classes qui ont une relation de dépendance :

Principe (utilisation de class loader)

Si un objet de type `C` est instancié dans `A`, alors `C` sera chargée via le *class loader* de `A`

Principe (espace de nommage)

Une classe ne peut par défaut qu'interagir avec les classes qui ont été chargées par le même *class loader*.

On peut donc avoir plusieurs classes différentes qui portent le même nom...

Le vérificateur de *bytecode*

Le vérificateur garantit que les classes chargées ont une structure interne correcte (en particulier les instructions de saut qui doivent « rester » dans la méthode).

4 passes :

1. au **chargement** du fichier, vérification de la structure du fichier *magic number*, bon nombre d'octets etc.)
2. au **link** de la classe, on vérifie dans un premier temps un certain nombre de propriétés ne dépendant pas du code à exécuter :
   - existence de la super-classe
   - attributs et méthodes avec des noms valides
   - respect de **`final`**
3. toujours au **link**, on analyse le code de chaque méthode :
   - les variables locales sont accédées si elles ont été initialisées
   - les méthodes sont invoquées avec les bons arguments
   - les attributs ont des valeurs du bon type
   - les *opcodes* ont les bons arguments
4. vérification des références symboliques à l'**exécution**, par exemple chargement des classes nécessaires.

Vérificateur de bytecode : exemple

Dans le bytecode de la classe `Personne`, méthode `getNom` :

`getfield #4` au lieu de `getfield #3`

Résultat :

```
Create object
Exception in thread "main" java.lang.VerifyError:
          (class: Personne, method: getNom signature: ()Ljava/lang/String;)
          Expecting to find object/array on stack
        at TestPersonne.main(TestPersonne.java:5)
```

Comprendre la classe ClassLoader

`ClassLoader` est une classe abstraite. Elle possède une méthode particulière à redéfinir :

```
protected Class findClass(String name) throws ClassNotFoundException
```

C'est cette méthode qui charge le *bytecode* de la classe.

Quelques méthodes utiles :

- `getClassLoader()` sur un objet de type `Class`
- `getSystemClassLoader()`
- `public class loadClass(String name)` qui fonctionne comme suit :
  - utilise `findLoadedClass` pour voir si la classe n'a pas été chargée précédemment
  - sinon on appelle `loadClass` sur le *class loader* père
  - si elle n'est toujours pas chargée, on utilise `findClass`
- `resolveClass(Class c)` qui *link* la classe

Comment utiliser un *class loader* ?

Chargement de la classe `Personne` :

```
Class personne = monCL.loadClass("Personne", true);
```

Création d'une instance de `Personne` :

```
Object o = personne.newInstance();
```

Utilisation de l'objet ?

```
((Personne) o).getName() // ne fonctionne pas par defaut !
```

Il faut une classe ou une interface en commun entre les deux *class loaders*.

Class loader : un exemple

Cesar.java

```
import java.io.*;

public class Cesar {
  public static void main(String[] args) {
    if (args.length != 3) {
      System.out.println("USAGE: java Cesar in out key");
      return;
    }

    try {
      FileInputStream in = new FileInputStream(args[0]);
      FileOutputStream out = new FileOutputStream(args[1]);
      int key = Integer.parseInt(args[2]);
      int ch;
      while ((ch = in.read()) != -1) {
        byte c = (byte)(ch + key);
            out.write(c);
      }
      in.close();
      out.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
```

Class loader : un exemple

ClassLoaderCesar.java

```
import java.io.*;

public class ClassLoaderCesar extends ClassLoader {

  private int key;

  public ClassLoaderCesar(int key_) {
    key = key_;
  }
```

Class loader : un exemple

ClassLoaderCesar.java

```
protected Class findClass(String name)
    throws ClassNotFoundException {
    byte[] classBytes = null;

    try {
       classBytes = loadClassBytes(name);
    } catch (IOException e) {
       throw new ClassNotFoundException(name);
    }

    Class cl = defineClass(name, classBytes, 0, classBytes.length);

    if (cl == null)
       throw new ClassNotFoundException(name);
    return cl;
}
```

Class loader : un exemple

ClassLoaderCesar.java

```java
  private byte[] loadClassBytes(String name)
      throws IOException {
    String cname = name.replace('.', '/') + ".cesar";
    FileInputStream in = null;
    in = new FileInputStream(cname);
    try {
      ByteArrayOutputStream buffer = new ByteArrayOutputStream();
      int ch;
      while ((ch = in.read()) != -1) {
        byte b = (byte) (ch - key);
        buffer.write(b);
      }
      in.close();
      return buffer.toByteArray();
    } finally {
      in.close();
    }
  }
}
```

Class loader : un exemple

ClassLoaderCesar.java

```
import java.lang.reflect.*;

public class TestClassLoaderCesar {
  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println("USAGE: java TestClassLoaderCesar classname key");
      return;
    }

    try {
      ClassLoader loader = new ClassLoaderCesar(Integer.parseInt(args[1]));
      Class c = loader.loadClass(args[0]);
      String[] callargs = new String[] {};

      Method m = c.getMethod("main", args.getClass());
      m.invoke(null, (Object) callargs);
    }
    catch (Throwable e) {
      e.printStackTrace();
    }
  }
}
```

Class loader : un exemple qui marche

ClassLoaderCesar.java

```
[tof@ordiflotof]...ransparents/JVM/src/cl % java TestClassLoaderCesar
                                                 TestPersonne 10
Create object
Object created
Christophe
[tof@ordiflotof]...ransparents/JVM/src/cl % java TestClassLoaderCesar
                                                 TestPersonne 8
java.lang.ClassFormatError: Incompatible magic value 3422600384 in class file Test
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClassCond(ClassLoader.java:632)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:616)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:466)
        at ClassLoaderCesar.findClass(ClassLoaderCesar.java:21)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:248)
        at TestClassLoaderCesar.main(TestClassLoaderCesar.java:12)
```

Références

Arnold, K., J. Gosling et D. Holmes (2005).
The Java Programming Language.
4e éd.
Java Series.
Addison-Wesley.

Wilson, S. et J. Kesselman (2001).
Java Platform Performance : Strategies and Tactics.
Available on http : / / java . sun . com / docs / books / performance/.
Prentice Hall.
http://java.sun.com/docs/books/performance/.