Programmation distribuée Java

Christophe Fouqueré

Master Informatique 1ère année

christophe.fouquere@univ-paris13.fr
A204, Université Paris 13

Very short bibliography :

- **Java La synthèse, Des concepts objet aux architectures Web**, Clavel et alii, Dunod Informatiques, 2000
- **Programmation réseau avec Java / Java Network Programming**, E. Rusty Harold, O'Reilly, 2001
- and of course `http://docs.oracle.com`

  in particular

  `http://docs.oracle.com/javase/10/docs/api/`

  or

  `https://docs.oracle.com/en/java/javase/13/`

1. Introduction
2. Threads : master-slave architectures
   - Threads / Lightweight processes
   - Typical examples
   - Runnable and Callable
   - Pools of threads
3. Network : adressing and sockets
   - Networking
     - Recalls on networks (IP, TCP, UDP, naming)
     - Class `InetAddress`
     - non-secured TCP Connections between hosts (class `Socket`)
     - Secured TCP Connections (class `SSLServerSocket`)
     - UDP Connections (class `DatagramSocket`)
     - Multicast Connections (class `MulticastSocket`)
   - URL
     - Classes `URL` and `URLConnection`
     - Management of protocols
   - RMI and RMI-IIOP
     - RMI : Remote Method Invocation
4. Annotations management

5. Tools : Messaging, name servers
   - Mailing
   - Name server

6. Client-server : control

7. Security policy in Java
   - Introduction
   - java.security
   - java.policy

1 Introduction

Objectives :

- Use of IDEs (Integrated Development Environments) as Eclipse, Netbeans
- Understanding of good practices in open environment : sockets, url, naming servers
- Applications : mail, rmi, . . .

- 1969 : Arpanet
- 1979 : Internet
- 1974-1982 : TCP
- 1991 : Web
- 1989-1996 : standard CORBA (Common Object Request Broker Architecture) from the OMG (Object Management Group)
- 1990-1996 : COM model (Component Object Model) then DCOM then ActiveX from Microsoft
- 1994 : Java
- 1997 : 1st "web service"
- 1998- : Development of a bunch of standard protocols (W3C, OMG, manufacturers)
- 2000- : Protocols Web2.0, BPEL, ...

Java : the basis !

Java : A class = structure + methods

```
package test;
import java.util.*;

public class Personne {
    private String nom;
    protected List<String> prenoms;
    public transient String societe;
    private static final String societeException
                                    = new String("inconnu");
    public boolean defini;
    public void identifiant() {System.out.println
                                    ("je␣m'appelle" + nom);}
    Personne(Object o) {
        if (o instanceof String) {this.nom = (String) o;}
    }

    public static void main(String args[]) {
        Personne p = new Personne("zz");
        assert p.nom == "t": "erreur␣de␣creation";
}    }
```

```
$ ls test
Personne.java
$ javac test/Personne.java
$ java test.Personne
```

```
$ java -ea test.Personne
Exception in thread "main" java.lang.AssertionError:
    erreur de creation
    at test.Personne.main(Personne.java:25)
```

```
package test;
import java.util.*;

public class Personne {
    private String nom;
    protected List<String> prenoms;
    public                 ing societe;
    private static final String societeException
                                    = new String("inconnu");
    public boolean defini;
    public void identifiant() {System.out.println
                                    ("je m'appelle" + nom);}
    Personne(Object o) {
        if (o instanceof String) {this.nom = (String) o;}
    }
        public static void main(String args[]) {
        Personne p = new Personne("zz");
        assert p.nom == "t": "erreur de creation";
}    }
```

package name (test.Personne)

class imports

class def (convention: 1st letter = capital)

class constructors

```
$ ls test
Personne.java
$ javac test/Personne.java
$ java test.Personne
```

```
$ java -ea test.Personne
Exception in thread "main" java.lang.AssertionError:
    erreur de creation
    at test.Personne.main(Personne.java:25)
```

```
package test;
import java.util.*;

public class Personne {
    private String nom;
    protected List<String> prenoms;
    public transient String societe;
    private static final String societeException
                                    = new String("inconnu");
    public boolean defini;
    public void identifiant() {System.out.println
                                    ("je m'appelle" + nom);}
    Personne(Object o) {
        if (o instanceof String) {this.nom = (String) o;}
    }

    public static void main(String args[]) {
        Personne p = new Personne("zz");
        assert p.nom == "t": "erreur de creation";
    }
}
```

locally usable

usable by heritage

globally usable

```
$ ls test
Personne.java
$ javac test/Personne.java
$ java test.Personne
```

```
$ java -ea test.Personne
Exception in thread "main" java.lang.AssertionError:
    erreur de creation
    at test.Personne.main(Personne.java:25)
```

```
package test;
import java.util.*;

public cla {
    private String nom;
    protected L st<Stri> prenoms;
    public transient Str societe;
    private static final String societeException
                                    = new String("inconnu");
    public boolean defini;
    publi ifiant() {System.out.println
                                    ("je_m'appelle" + nom);}
    Personne(Object o) {
        if (o instanceof String) {this.nom = (String) o;}
    }

    public st in(String args[]) {
        Personne p = new Personne("zz");
        assert p.nom == "t": "erreur_de_creation";
}    }
```

non serializable

immutable after instanciation

class variable

test

```
$ ls test
Personne.java
$ javac test/Personne.java
$ java test.Personne
```

```
$ java -ea test.Personne
Exception in thread "main" java.lang.AssertionError:
    erreur de creation
    at test.Personne.main(Personne.java:25)
```

```
package test;
import java.util.*;

public class Personne {
    private String nom;
    protected List<String> prenoms;
    public transient String societe;
    private static final String societeException
                                    = new String("inconnu");
    public boolean defini;
    public void identifiant() {System.out.println
                                ("je m'appelle" + nom);}

    Personne(Object o) {
        if (o                              s.nom = (String) o;}
    }

    public static void main(String args[]) {
        Personne p = new Personne("zz");
        assert p.nom == "t": "erreur de creation";
}     }
```

parameterized type

primitive type (byte, char, double, float, int, long, short)

main method for a program

assertion / exception

```
$ ls test
Personne.java
$ javac test/Personne.java
$ java test.Personne
```

```
$ java -ea test.Personne
Exception in thread "main" java.lang.AssertionError:
    erreur de creation
    at test.Personne.main(Personne.java:25)
```

Java : Recalls : interfaces

```
public interface DigestListener {
   public void setDigest(byte[] digest);
}
```

```
import java.io.*;

public class DigestServer implements DigestListener {
   ...
   public void setDigest(byte[] digest) {
      ...
   }
   ...
}
```

Java : Recalls : interfaces

interface = type of a (part of a) class

```
public interface DigestListener {
    public void setDigest(byte[] digest);
}
```

```
import java.io.*;

public class DigestServer implements DigestListener {
    ...
    public void setDigest(byte[] diges
        ...
    }
    ...
}
```

class implementing the methods of the interface

Java : Recalls : abstract classes and extensions

```
abstract class Triangle {
  int[] cotes = new int[3];
  abstract double surface();
  public String toString(){
    return "Triangle␣de␣cotés␣" + cotes[0] + ",␣"
                + cotes[1] + ",␣" + cotes[2]
                + "␣et␣de␣surface␣" + surface();
  }
}
```

```
public class Isocele extends Triangle {
  double surface(){
    return (cotes[0]
        *java.lang.Math.sqrt(cotes[1]*cotes[1]
                             -cotes[0]*cotes[0]/4)/2);
  }
  Isocele(int a, int b, int c) {
    cotes[0] = a; cotes[1] = b; cotes[2] = c;
  }
  public static void main(String args[]) {
    Triangle t = new Isocele(2,3,3);
    System.out.println(t);
  }
}
```

Java : Recalls : abstract classes and extensions

```
abstract class Triangle {
  int[] cotes = new int[3];
  abstract double surface();
  public String toString(){
    return "Triangle de cotés " + cotes[0] + ", "
                + cotes[1] + ", " + cotes[2]
                + " et de surface " + surface();
  }
}
```

abstract class, abstract method

class extension

```
public class Isocele extends Triangle {
  double surface(){
    return (cotes[0]
        *java.lang.Math.sqrt(cotes[1]*cotes[1]
                             -cotes[0]*cotes[0]/4)/2);
  }
  Isocele(int a, int b, int c) {
    cotes[0] = a; cotes[1] = b; cotes[2] = c;
  }
  public static void main(String args[]) {
    Triangle t = new Isocele(2,3,3);
    System.out.println(t);
  }
}
```

Java : Recalls : abstract classes and extensions

```
abstract class Triangle {
  int[] cotes = new int[3];
  abstract double surface();
  public String toString(){
    return "Triangle de cotés " + cotes[0] + ", "
              + cotes[1] + ", " + cotes[2]
              + " et de surface " + surface();
  }
}
```

method overloading

```
public class Isocele extends Triangle {
  double surface(){
    return (cotes[0]
        *java.lang.Math.sqrt(cotes[1]*cotes[1]
                             -cotes[0]*cotes[0]/4)/2);
  }
  Isocele(int a, int b, int c) {
    cotes[0] = a; cotes[1] = b; cotes[2] = c;
  }
  public static void main(String args[]) {
    Triangle t = new Isocele(2,3,3);
    System.out.println(t);
  }
}
```

Java : Recalls : exceptions

```
class ExpandableArray {
   protected Object[] data;
   protected int size = 0;

   public ExpandableArray(int cap) {data = new Object[cap];}
   public int size() { return size;}

   public Object get(int i) throws NoSuchElementException {
      if (i<0 || i>= size) throw new NoSuchElementException();
      return data[i];
   }
   public void add(Object x) {
      if (size == data.length) {
         Object[] olddata = data;
         data = new Object[3 * (size + 1) / 2];
         System.arraycopy( olddata, 0, data, 0, olddata.length);
      }
      data[size++] = x;
}  }

class NoSuchElementException extends Exception {};
```

Java : Recalls : exceptions

```
class ExpandableArray {
   protected Object[] data;
   protected int size = 0;

   public ray(int cap) {data = new Object[cap];}
   public int size() { return size;}

   public Object get(int i) throws NoSuchElementException {
      if (i<0 || i>= size) throw new NoSuchElementException();
      return data[i];
   }
   public void add(Object x) {
      if (size == data.length) {
         Object[] olddata = data;
         data = new Object[3 * (size + 1) / 2];
         System.arraycopy( olddata, 0, data, 0, olddata.length);
      }
      data[size++] = x;
}  }

class NoSuchElementException extends Exception {};
```

to put and treat exceptions

```
import java.io.*; import java.net.Socket;

public class Main {
  public static void main(String[] args) {
    Socket connexion = null;
    try {
      connexion = new Socket("www.univ-paris8.fr",80);
      Writer output = new OutputStreamWriter
                          (connexion.getOutputStream(), "8859_1");

      output.write("GET/HTTP␣1.1\nHost:www.univ-paris8.fr\n\n");
      output.flush();

      BufferedReader input =
        new  BufferedReader(
          new InputStreamReader
                    (connexion.getInputStream(),"8859_1"),1024);

      StringBuffer sb = new StringBuffer(); int c;
      while ((c = input.read()) != -1) sb.append((char) c);
      System.out.println(sb);
      System.out.flush();
    }
    catch (IOException e) {System.out.println(e);}
    finally {
      try {if (connexion != null) connexion.close();}
      catch (IOException e) {System.out.println(e);}
} } }
```

```
import java.io.*; import java.net.Socket;

public class Main {
  public static void main(String[] args) {
    Socket connexion = null;
    try {
      connexion = new Socket("www.univ-paris8.fr",80);
      Writer output = new OutputStreamWriter
                          (connexion.getOutputStream(), "8859_1");

      output.write("GET/HTTP␣1.1\nHost:www.univ-paris8.fr\n\n");
      output.flush();

      BufferedReader input =
        new  BufferedReader(
          new InputStreamReader
                    (connexion.getInputStream(),"8859_1"),1024);

      StringBuffer sb = new StringBuffer(); int c;
      while ((c = input.read()) != -1) sb.append((char) c);
      System.out.println(sb);
      System.out.flush();
    }
    catch (IOException e) {System.out.println(e);}
    finally {
      try {if (connexion != null) connexion.close();}
      catch (IOException e) {System.out.println(e);}
} } }
```

test and local treatment

Writing a program

- 1 kind of object = 1 class
- 1 project = 1 executable = 1 set of classes (divided into packages)
- "ordinary" software => 1 project
- client-server software => 2 projects
- 1 project =
  - 1 class `Main` (the main initializer)
  - 1 class per kind of object

1 class = fields + constructors + functionalities

- Defining 1 class =
  - define the fields (private)
    - read/write control on field values
    - easier for logging, security, ...
  - add getters and setters
    - To be systematically used !
  - add constructors
  - add one method per functionality

```
public class Enseignant {

    private String nom;
    private int age;
    private int salaire;

    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
   ...
    Enseignant(String nom, int age, int salaire, Grade grade) {
        this.nom = nom;
        this.age = age;
        this.salaire = salaire;
        this.grade = grade;
    }
   ...
   @Override
    public String toString() {
        return getNom() + "␣" + getAge() + "ans";
    }
```

Case of lists of objects of a class :

- Either define a special class : necessary when several different lists for the same class

```
public class ListeEnseignant {
    ...
     private List<Enseignant> personnel = new ArrayList();
    ...
     public List<Enseignant> getPersonnel() {
         return personnel;
     }

     public void addEnseignant(Enseignant enseignant) {
         getPersonnel().add(enseignant);
     }
     public void delEnseignant(Enseignant enseignant) {
         getPersonnel().del(enseignant);
     }
     ...
}
```

- Or use a static field :

```
public class Enseignant {
    ...
     private static List<Enseignant> personnel = new ArrayList();
    ...
     public static List<Enseignant> getPersonnel() {
         return personnel;
     }

     public static void addEnseignant(Enseignant enseignant) {
         Enseignant.personnel.add(enseignant);
     }
     public static void delEnseignant(Enseignant enseignant) {
         Enseignant.personnel.del(enseignant);
     }
    ...
}
```

Java : Version 1.4

- Assertions :

```
if (i % 3 == 0) {
    ...
} else if (i % 3 == 1) {
    ...
} else { // We know (i ...
}
```

becomes

```
if (i % 3 == 0) {
   ...
} else if (i % 3 == 1) {
    ...
} else {
    assert i % 3 == 2 : i;
    ...
}
```

Java : Version 1.5

Generic types :

```
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
      if (((String) i.next()).length() == 4)
        i.remove();
}
```

becomes

```
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
      if (i.next().length() == 4)
        i.remove();
}
```

Java : Version 1.5

Iterations :

```
int sum(int[] a) {
    int result = 0;
    for (int i : a)
        result += i;
    return result;
}
```

Java : Version 1.5

Iterations :

The variable *i* goes through *a*

```
int sum(int[] a) {
    int result = 0;
    for (int i : a)
        result += i;
    return result;
}
```

Java : Version 1.5

- Boxing (i.e. automatic change between primitive types and their associated classes, e.g. `int` and `Integer`).
- Variable number of arguments :

```
public static String format(String pattern,
                            Object... arguments);
```

- Enumeration Type :

```
public enum Saison { PRINTEMPS, ETE, AUTOMNE, HIVER }
for (Saison saison : Saison.values())
      System.out.println(saison);
```

Java : Version 1.5

- Enumeration type :

```
public enum Grade {
    PU ("PU", "1"),
    MCU ("mcu", "2"),
    PRAG ("PRAG", "3"),
    BIATTS ("BIATTS", "4"),
    PUPH ("PUPH", "5"),
    MCUPH ("MCUPH", "6");

    private final String i;
    private final String grade;

    Grade(String i, String grade) {
        this.i=i;
        this.grade=grade;
    }
}
public static void main(String[] args) {
     Grade gr = Grade.valueOf("MCU");
    Grade gr1 = Grade.MCU;
     System.out.println(gr1.grade);
}
```

Java : Version 1.5

- Stubs for RMI dynamically generated (no more need for `rmic`)
- Annotations : allows intermediary softwares (compilers, interpreters, environnements, ...) to test, verify or even add code.

Declaring an annotation :

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Java : Version 1.5

Putting annotations in a code :

```
public class Foo {
    @Test public static void m1() { }
    public static void m2() { }
    @Test public static void m3() {
        throw new RuntimeException("Boom");
    }
    public static void m4() { }
    @Test public static void m5() { }
    public static void m6() { }
    @Test public static void m7() {
        throw new RuntimeException("Crash");
    }
    public static void m8() { }
}
```

Java : Version 1.5

Defining code that uses the annotation (during compilation of the program or its execution) :

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

public class RunTests {
   public static void main(String[] args) throws Exception {
      int passed = 0, failed = 0;
      for (Method m : Class.forName(args[0]).getMethods()) {
         if (m.isAnnotationPresent(Test.class)) {
            try {
               m.invoke(null);
               passed++;
            } catch (Throwable ex) {
               System.out.printf("Test␣%s␣failed:␣%s␣%n",
                                 m, ex.getCause());
               failed++;
            }
         }
      }
      System.out.printf("Passed:␣%d,␣Failed␣%d%n", passed,
                                                   failed);
   }
}
```

Java : Version 1.6

New or improved libraries :

- JAX-WS : web services
- JDBC : API to data bases
- Java compiler API
- New predefined annotations
- Scripting

Java : Version 1.7

Minor modifications :

```
List<String> list = new ArrayList<String>();

Map<Reference<Object>,Map<String,List<Object>>> map =
     new HashMap<Reference<Object>,Map<String,List<Object>>>();
```

versus

```
List<String> list = new ArrayList<>();

Map<Reference<Object>,Map<String,List<Object>>> map =
     new HashMap<>();
```

and also pointers to functions, ... (cf. `java.lang.invoke`)

Java : Version 1.8

- Lambda expressions
- java.util.stream
- security, ...

Java : Version 1.8

Syntax of a lambda-expression :

```
(Type1 var1,...,Typep varp) -> corps
```

```
() -> System.out.println(this)

(String str) -> System.out.println(str)

str -> System.out.println(str)

(String s1, String s2) -> { return s2.length() - s1.length(); }

Arrays.sort(strArray,
   (s1, s2) -> s2.length() - s1.length());
```

Possibility to use extern variables in the body or arguments if these variables are unchanged in the following (kind of **`static`**).

Java : Version 1.8

The type of a lambda-expression is a **functional interface** : interface with a unique method (quite ...) :

```
@FunctionalInterface
public interface Somme {
    public int somme(int n1, int n2);
}
```

Use in a code :

```
public class Test {
    public static void main(String[] args) {
        Somme somme =(int a, int b) -> a+b;
        int resultat = somme.somme(3, 4);
        System.out.println(resultat);
    }
}
```

Java : Version 1.8

Facility to access directly to the methods :

```
import java.io.*;
import java.util.*;

class Test {

  public static void main(String[] args) {
     PolygoneRegulier p1 = new PolygoneRegulier(2,4);
     PolygoneRegulier p2 = new PolygoneRegulier(3,6);
     PolygoneRegulier p3 = new PolygoneRegulier(2,8);
     PolygoneRegulier p4 = new PolygoneRegulier(4,4);
     List<PolygoneRegulier> lp = Arrays.asList(p1,p2,p3,p4);

     lp.stream()
           // filtrage
           .filter(x -> x.getNbCotes() == 2)
           // mapping
           .map(x -> {x.setNbCotes(x.nbCotes+3); return x;})
           .forEach( System.out::println );
  }
}
```

(following slide for `PolygoneRegulier`)

```
class PolygoneRegulier {
   int nbCotes, lgCotes;

   PolygoneRegulier(int nbCotes,int lgCotes) {
     setNbCotes(nbCotes);
     setLgCotes(lgCotes);
   }

   public String toString() {
      return "Polygone␣Régulier:␣nbCotes␣=␣" + nbCotes
                + ",␣lgCotes␣=␣" + lgCotes;
   }

   public void setNbCotes(int nbCotes) {
      this.nbCotes = nbCotes;
   }
   public void setLgCotes(int lgCotes) {
      this.lgCotes = lgCotes;
   }
   public int getNbCotes() {
      return nbCotes;
   }
   public int getLgCotes() {
      return lgCotes;
   }
}
```

Java : Version 1.8

A `Stream` is a class with methods able to manage streams of objects.
A stream may be created from collections, lists, ... with the method `stream()`.
A stream allows for modifying objects on the fly. Modifiers can be iterated, or can be applied to concatenate results.
Functional arguments have the following annotation : `FunctionalInterface` together with one of the following types (or a variant) :

- `Function<T,R>` : takes an object of type T and returns an object of type R
- `Supplier<R>` : returns an object of type R
- `Predicate<T>` : returns a boolean wrt the type T
- `Consumer<T>` : does an operation on an object of type T and "consumes" it.

Java : Version 1.8

Example with `Consumer<T>` :

```
import java.util.*;
import java.util.function.Consumer;
public class Test {
   public static void main(String[] args) {
      List<Personne> list = new ArrayList<Personne>();
      list.add(new Personne("Younes",45,"PR"));
      list.add(new Personne("Jean-Yves",40,"MCF"));

      Consumer<Personne> impression
          = (Personne p) -> System.out.println
                              ("Nom : "+p.nom
                               +", Age : "+p.age
                               +", Profession : "+p.profession);

      //3 possibilites :
      list.forEach(impression);
      list.forEach(Personne::impression);
      list.forEach(p -> p.impression());
   }
}
```

together with the class Personne (following slide)

```
public class Personne {
    public String nom;
    public int age;
    public String profession;
    public Personne(String nom,int age,String profession){
        this.nom = nom;
        this.age = age;
        this.profession = profession;
    }
    public void impression(){
        System.out.println("Nom␣:"+nom
                           +",␣Age␣:"+age
                           +",␣Profession␣:"+profession);
    }
}
```

Another example with `Predicate` and 1 parallel stream :

```
import java.util.*;
import java.util.function.Predicate;
public class Test {
   public static void main(String[] args) {
      List<Personne> list = new ArrayList<Personne>();
      list.add(new Personne("Younes",50,"PR"));
      list.add(new Personne("Jean-Yves",40,"MCF"));
      list.add(new Personne("Christophe",53,"PR"));

      Predicate<Personne> isProfesseur
                  = e -> e.profession.equals("PR");
      OptionalDouble ageMoyenPr = list
          .parallelStream()
          .filter(isProfesseur)
          .mapToDouble(e -> e.age)
          .average();

      System.out.println(ageMoyenPr.getAsDouble());
   }
}
```

Another example with `Predicate` and 1 sequential stream :

```
import java.util.*;
import java.util.function.Predicate;
public class Test {
    public static void main(String[] args) {
      List<Personne> list = new ArrayList<Personne>();
      list.add(new Personne("Younes",50,"PR"));
      list.add(new Personne("Jean-Yves",40,"MCF"));
      list.add(new Personne("Christophe",53,"PR"));

      Predicate<Personne> isProfesseur
                    = e -> e.profession.equals("PR");
      int nombre = list
          .stream()
          .filter(isProfesseur)
          .mapToInt(e -> 1)
          .sum();

      System.out.println(nombre);
    }
}
```

Java : Version 9

News :

- `jshell` : top-level interpreter of Java code
- Introduction of **modules** :

```
// file module-info.java compiled by javac
module monModule{
   requires module1;
   exports monModule.Classe; // for all
   opens monModule.ClasseBis to autreModule; // restricted
}
```

Java : Version 10

News :

- Introduction of **var** :

```
var list = new ArrayList<String>();
// instead of
ArrayList<String> list = new ArrayList<String>();

var bytes = Files.readAllBytes(path);
// instead of
bytes[] bytes = Files.readAllBytes(path);

for (var counter=0; counter<10; counter++)  {...}
// instead of
for (int counter=0; counter<10; counter++)  {...}
```

Java : Version 11

Nothing special !
Note :

- new version every 6 months
- new version = cancelling of bugs

2 Threads : master-slave architectures

- Threads / Lightweight processes
- Typical examples
- Runnable and Callable
- Pools of threads

Package : `java.lang.Thread`
(local) distribution (locale) of computations :
1 task done by one computing device with

total autonomy of memory	==>	processus
sharing of the memory	==>	thread

Thread :

creation	==>	`thread()`
initialization phase	==>	`start()`
computing phase	==>	`run()`
stop phase	==>	`interrupt()` (or `stop()` if redefined)
freeing phase	==>	done by the garbage collector

Shared memory between threads :

- **Take care of variables** :
  - variables with values synchronized between threads (otherwise in cache)

    ==> **`volatile`**
- **during computation phase** :
  - waiting for values

    ==> `interrupt() / join() / sleep()`
  - synchronization between threads (e.g. read/write on shared values)

    ==> **`synchronized`**

Classes `Thread` and `Runnable` :

```
class T extends Thread {
    ...
    public void run() {
        ...
    }
    ...
}
...
Thread t = new T(...);
t.start();
...
```

```
class T implements Runnable {
    ...
    public void run() {
        ...
    }
    ...
}
...
T tSpec = new T(...);
Thread t = new Thread(tSpec);
t.start();
...
```

- launching a process : a unique thread starts executing `main()`
- class `Thread`
  - **`static`** `Thread currentThread()`
    returns the current thread (i.e. the instance)
  - **`void`** `setName(String s)`
    gives a name to a Thread (used by `toString`) (by default `Thread-i` and `Thread-0` for the main one)
- The call to `start()` creates really a thread (in terms of operating systems) and calls `run()`.
- A call to `run()` executes only the method `run()` without creating a new thread.

```
class T implements Runnable{
   public void run() {
      for (int i = 0; i<5; i++) {
         System.out.println("Processus léger T" + i);
         try { Thread.sleep(500);}  // 500 msec = 1/2 s
         catch (InterruptedException e) {
            System.out.println("Interruption");
      }  }
      System.out.println("Processus léger T terminé");
}  }

public class TTest{
   public static void main(String[] args) throws Exception {
      Thread t = new Thread(new T());
      t.start();

      for (int i = 0; i<5; i++) {
         System.out.println("Processus léger main" + i);
         Thread.sleep(500);
      }
      System.out.println("Processus léger main terminé");
}  }
```

- **Priority** :
  - between `Thread.MIN_PRIORITY=1` and `Thread.MAX_PRIORITY=10`
  - by default `Thread.NORM_PRIORITY=5`
  - class `Thread`
    - **`int`** `getPriority()` returns the priority of the Thread
    - **`void`** `setPriority(`**`int`** `priority)` sets the priority

- **Synchronization** (only one thread at one time). To each object is associated a descriptor containing informations on that object. The synchronization goes like this :
  - a specific information (a flag) signals if this object is already in use.
  - The execution of an instruction (or a block, or a method) may be conditioned by this flag

```
...
public void maMéthode(C a, ...) {
    ...
    synchronized(o) {
// flag sur o
        ... o.f(...) ...
    }
    ...
}
...
```

- `this` may be used for synchronization (implicit when)
- methos may be used for synchronization except :
  - methods of interface
  - constructors
- the specification of a synchronization is not inheritable
- synchronization of static fields is doable on objects of type class (`... synchronized(C.class) ...`)

```
class ExpandableArray {
   protected Object[] data; protected int size = 0;

   public ExpandableArray(int cap) {data = new Object[cap];}
   public synchronized int size() { return size;}

   public synchronized Object get(int i)
   throws NoSuchElementException {
      if (i<0 || i>= size) throw new NoSuchElementException();
      return data[i];
   }
   public synchronized void add(Object x) {
      if (size == data.length) {
         Object[] olddata = data;
         data = new Object[3 * (size + 1) / 2];
         System.arraycopy( olddata, 0,
                           data, 0, olddata.length);
      }
      data[size++] = x;
   }
   public synchronized void removeLast()
   throws NoSuchElementException {
      if (size == 0) throw new NoSuchElementException();
      data[--size] = null;
}  }

class NoSuchElementException extends Exception {};
```

Management of shared/local memory

- each instance (of `Runnable`) has its own variables
- threads launched on the same instance share these variables
- the modifier **`volatile`** forces synchronization of shared variables

Exemple :

```
public class TestRunnable implements Runnable { volatile int x;
   public TestRunnable(int x) {this.x = x;}
   public void run() {
      System.out.print(x++ + "␣" + this);
      try {Thread.sleep(2000);}
      catch (InterruptedException e) {System.err.println(e);};
      System.out.print(x + "␣" + this);
   }
   public static void main(String args[]) {
      TestRunnable r = new TestRunnable(0);
      TestRunnable s = new TestRunnable(10);
      Thread tr1 = new Thread(r); Thread tr2 = new Thread(r);
      Thread ts = new Thread(s);
      tr1.start(); tr2.start();ts.start();
}  }
```

Results of execution :

```
0 1 TestRunnable@119298d
10 TestRunnable@119298d
TestRunnable@f72617
2 TestRunnable@119298d
2 TestRunnable@119298d
11 TestRunnable@f72617
```

Local data : class `ThreadLocal`

- `set(Object o)`
- `get(Object o)`

Example :

```
public class R implements Runnable {
   Object o;       // shared variable

   ThreadLocal v = new ThreadLocal();
                    // local variable
   ...
      o = ...;
      v.set(u);         // where u is some object
      ...
      .. = .. o ..;
      .. = .. v.get() ..;
}
```

Group of threads

- class `ThreadGroup` (by default, the group of a thread is its creator's group) :
  - `ThreadGroup ThreadGroup(String s)` creates a threadgroupe with name `s`
  - **`void`** `setMaxPriority(`**`int`**` priority)` sets the maximal priorityof a threadgroup
- class `Thread` :
  - `ThreadGroup getThreadGroup()` returns the threadgroup of the thread
  - `Thread Thread(ThreadGroup g, ImplRun r, String s)` creates a new thread in the group `g` with Runnable `r` (`ImplRun` should implement `Runnable`) and name `s`

1. Callback after thread call (continuation method)

```
// Slave
import java.io.*; import java.security.*;

public class CbDigestSlave implements Runnable {
   private File input;                          // file to be signed
   private CbDigestMaster cbDigestMaster;       // caller object

   public CbDigestSlave (File input, CbDigestMaster cb) {
      this.input = input;
      this.cbDigestMaster = cb;
   }
...
```

```
...
   public void run() {
      try {
         MessageDigest sha = MessageDigest.getInstance("SHA");
         DigestInputStream din = new DigestInputStream(
            new FileInputStream(input),
            sha
         );
         int b;
         while ((b = din.read()) != -1);
         din.close();
         byte[] digest = sha.digest();
         cbDigestMaster.setDigest(digest);   // continuation
      }
      catch(IOException e) {System.err.println(e);}
      catch(NoSuchAlgorithmException  e) {System.err.println(e);}
}  }
```

```java
// Master
import java.io.*;

public class CbDigestMaster {
   private File input;
   private byte[] digest;

   public CbDigestMaster(File input) {
      this.input = input;
   }

   public void calculateDigest() {
      CbDigestSlave cb = new CbDigestSlave(input, this);
                                          // this = continuation
      Thread t = new Thread(cb);       // thread call
      t.start();
   }

   void setDigest(byte[] digest) {        // receiving method
      this.digest = digest;
      System.out.println(this);
   }
... // see next slide
```

```
...
   public String toString() {
      String result = input.getName() + ": ";
      if (digest != null) {
         for (int i = 0; i< digest.length; i++) {
            result += digest[i] + " ";
         }
      } else {
         result += "unusable digest";
      }
      return result;
   }

   public static void main(String args[]) {
      for (int i = 0; i< args.length; i++) {
         File f = new File(args[i]);
         CbDigestMaster d = new CbDigestMaster(f);
         d.calculateDigest();
      }
   }
}
```

2. Callback after thread call (listeners)

```
// Slave
import java.util.*; import java.io.*; import java.security.*;

public class CbDigestSlave implements Runnable {
   private File input;
   private Vector digestList = new Vector(); // list of subscribers

   public CbDigestSlave (File input) { this.input = input; }

   public synchronized void addListener(DigestListener dl) {
      digestList.add(dl);
   }
   public synchronized void removeListener(DigestListener dl){
      digestList.remove(dl);
   }
   private synchronized void sendDigest(byte[] digest) {
      ListIterator iterator = digestList.listIterator();
      while(iterator.hasNext()) {
         DigestListener dl = (DigestListener) iterator.next();
         dl.setDigest(digest);
   }  }
   public void run() {
      ...
      sendDigest(digest);      // broadcasting
      ...
}  }
```

```java
// Interface Listener

public interface DigestListener {
   public void setDigest(byte[] digest);
}
```

```java
// Master

import java.io.*;


public class CbDigestMaster implements DigestListener {
   ...
   public void setDigest(byte[] digest) { // receiving method
      ...
   }
   ...
}
```

3. Synchronization between threads : `join()`

```
// Slave
public class CbDigestSlave extends Thread {
    ...
    public byte[] getDigest() {
        ...
    }
    ...
}
```

```java
// Master
import java.io.*;

public class CbDigestMaster {
   public static void main(String args[]) {
      CbDigestSlave[] cbDigestTab = new CbDigestSlave[args.length];
      int i;

      for (i=0;i<args.length;i++) {
         File f = new File(args[i]);
         cbDigestTab[i] = new CbDigestSlave(f);
         cbDigestTab[i].start();          // threads launching
      }

      for (i=0;i<args.length;i++) {
         try {
            cbDigestTab[i].join(); // freezes until thread i ends
            byte[] digest = cbDigestTab[i].getDigest();
            String fileName = cbDigestTab[i].getFileName();
            System.out.print(fileName + ":␣");
            ...
}  }  }  }
```

4. Thread waits : `wait()`

Remarques :

- `wait()` and `notify()` are Object methods
- `wait()` causes a wait to lift the lock
- `wait()` inside **`synchronized`**
- Variants for `wait()` :
  - `wait()`
  - `wait(long millisec)`
  - `wait(long millisec, int nanosec)`
- Variants for `notify()` :
  - `notify()` thread awakening
  - `notifyAll()` threads awakening

```
// Slave
import java.io.*; import java.security.*;

public class CbDigestSlave implements Runnable {
   private File input;
   private byte[] digest;
   private CbDigestMaster master; // notifier

   public CbDigestSlave (File input, CbDigestMaster master) {
      this.input = input;
      this.master = master;
   }

   public void run() {
    synchronized (master) {
      try {
         ...
         master.setDigest(digest);
         master.notify();          // notification
      }
      ...
    }
}  }
```

```
// Master
import java.io.*;

public class CbDigestMaster {
   private File input; private byte[] digest;

   public CbDigestMaster(File input) {
      this.input = input;
   }
   public void setDigest(byte[] digest) {
      this.digest = digest;
   }
   ...
   public void calculateDigest() {
    synchronized(this) {
      CbDigestSlave cb = new CbDigestSlave(input, this);
                                    // this = waiting object
      Thread t = new Thread(cb);    // thread call
      t.start();
      try {
         wait();
      }
      catch (InterruptedException e) {
      }
      System.out.println(this);      // digest known
   }}
...
}
```

2 Threads : master-slave architectures

- Threads / Lightweight processes
- Typical examples
- Runnable and Callable
- Pools of threads

Java : Version 1.5 / Runnable and Callable Interfaces

Similarities and differences :

- `java.lang.Runnable` :
  - Classes implementing `Runnable` should define a method `void run()`
  - a runnable instance is executed by a thread by means of a call to `start` (hence `run`)
  - No output
  - No exception thrown
- `java.util.concurrent.Callable` :
  - Classes implementing `Callable<V>` should define a method `V call()`
  - a callable instance is executed by a thread by means of a call to `call`
  - The result is of type `V` (type parameter)
  - Exceptions may be thrown

Java : Version 1.5 / Runnable and Callable Interfaces

Examples :

```
import java.io.*;

public class R implements Runnable {
    public void run()
    {
        System.out.println("Hello World!");
    }
}
```

```
import java.io.*;
import java.util.concurrent.*;

public class C implements Callable {
    public Integer call()
    {
        System.out.println("Hello World!");
        return 1;
    }
}
```

Java : Version 1.5 / Runnable and Callable Interfaces

Two ways to launch a Runnable via a thread :

```
import java.io.*;

public class Test {
  public static void main(String args[]) {
   R r = new R();
   Thread th = new Thread(r);
   th.start();
} }
```

```
import java.io.*;
import java.util.concurrent.*;

public class Test {
  public static void main(String args[]) {
   R r = new R();
   ExecutorService es = Executors.newSingleThreadExecutor();
   Future<?> futureResult = es.submit(r);
   es.shutdown();
} }
```

Java : Version 1.5 / Runnable and Callable Interfaces

One way to launch a Callable via a thread :

```
import java.io.*;
import java.util.concurrent.*;

public class Test {
  public static void main(String args[]) {

   C c = new C();

   ExecutorService es = Executors.newSingleThreadExecutor();
   Future<Integer> futureResult = es.submit(c);
   try {
      System.out.println(futureResult.get());
   }
   catch (Exception e) { System.out.println(e);};
   es.shutdown();
  }
}
```

Java : Version 8 / Runnable, Callable and lambdas

`Runnable` and `Callable` are annotated `@FunctionalInterface` :

```
Runnable r = () -> System.out.println("Hello World!");
Thread th = new Thread(r);
th.start();
```

```
Callable<Integer> r = () ->
    {
        System.out.println("Hello World!");
        return 1;
    }
Thread th = new Thread(r);
th.start();
```

2 Threads : master-slave architectures

- Threads / Lightweight processes
- Typical examples
- Runnable and Callable
- Pools of threads

Java : Version 1.5 / pools of threads or processes

In the package `java.util.concurrent` :

1) class `Executors` : in particular, creates a pool of threads :

- Use on the fly of free threads from the pool
- `static ExecutorService newSingleThreadExecutor()` : Creation of a pool of only one thread, with a stack of demands of "unlimited" size
- `static ExecutorService newFixedThreadPool(int n)` : Creation of a pool of `n` threads

Java : Version 1.5 / pools of threads or processes

In the package `java.util.concurrent` :

Useful with client-server systems ! (hence sockets, see next chapter)

2) interface `ExecutorService` :

- In particular, allows for launching executions, either `Callable` or `Runnable`
- **`void`** `shutdown()` Finishes works in progress and stops the pool
- **`void`** `execute(Runnable command)` Asks for the execution of the command by one of the threads of the pool
- `Future<T> submit(Callable<T> task)` Asks for the execution of the command by one of the threads of the pool and returns an instance of `Future`

Java : Version 1.5 / pools of threads or processes

In the package `java.util.concurrent` :

3) Interface `Future<T>` :

- Pointer to an object (asynchronous execution)
- `boolean cancel(boolean mayInterruptIfRunning)` Attempts to cancel execution of this task.
- `T get()` Waits and returns the result
- `T get(long timeout, TimeUnit unit)` Waits for `timeout` time units and returns the result (if time is over, returns an exception)

Java : Version 1.5 / pools of threads or processes

In the package `java.util.concurrent` :

4) Interface `CompletionService<V>` :

- Similar to `ExecutorService` but no need for waiting future by future : retrieves the first result as soon as it exists
- `Future<T> submit(Callable<T> task)` Asks for the execution of the command by one of the threads of the pool and returns an instance of `Future`
- `Future<T> take()` Waits and returns the first completed future

Implemented by class `ExecutorCompletionService` :

```
ExecutorService es = Executors.newFixedThreadPool(10);
CompletionService<Integer> cs =
            new ExecutorCompletionService<Integer>(es);
```

3 Network : adressing and sockets

- Networking
- URL
- RMI and RMI-IIOP

Recalls on networks

- **IP** : *Internet Protocol*
  - the only network protocol recognized by Java (so no IPX and no Appletalk)
  - data transmitted by datagrams (header + data)
  - address IPv4 with 4 bytes (IPv6 with 16 bytes)
- **DNS** : *Domain Name System*
  - translation symbolic name - IP address
  - Ex. (cf. `nslookup`) : `www.univ-paris13.fr` 192.33.182.1
- **TCP** : *Transmission Control Protocol*
  - packets rebuilt at the end + ack
- **UDP** : *User Datagram Protocol*
  - neither verification of data, nor order garranty
- **port** : (0 < port < 65535)
  - service determined as 'IP address + port'
  - Examples (cf /etc/services) :
    - 21 ftp
    - 25 smtp
    - 80 HTTP (web par défaut)
    - 1099 RMI registry

- **URI** : *Uniform resource Identifier*
  - resource address
  - *schema proper to the service*
  - in general : `schema://authority/path?request` where
    - *authority* = receiver address
    - *path* = path "in" this address (may be reinterpreted by the authority)
    - *request* = data or request
- **URN** : *Uniform Resource Name*
  - Example : `urn:isbn:1234567890`
  - data management by naming domain,
  - data retrieval from a server
- **URL** : *Uniform Resource Locator*
  - of general form

```
protocol://login:passwd@authority:port/path#section?request
```

- **file**://<host>/<path> — file on a local disk
  - file:///etc/services
  - <host> — by default local computer
- **ftp**://<user>:<pwd>@<machine>:<port>/path;type=<typecode>
  - ftp://ftp.univ-mrs.fr/f.txt — remote file (transfer)
  - <user> — by default anonymous
  - <pwd> — by default, email
  - <port> — by default, 21
  - <typecode> :
    - d listing of a folder
    - a ASCII transfer
    - i binary transfer
- **http**://<machine>:<port>/<path>?<request>
  - http://www.univ-paris13.fr (by default completed with index.html)
- **mailto**:<emailadress> — mail sender
  - mailto:christophe.fouquere@univ-paris13.fr
- **imap**://<user>:<passwd>@<host>/<path> — mail server
  - imap://login@hostAddress/folder
  - pop3://login@hostAddress/folder
- **telnet**://<user>:<passwd>@<host>:<port>/
  - telnet://F205-3/ — telnet connexion

(exception : `UnknownHostException`)

IP and URL Addresses

An instance of `InetAddress` contains the following information :

- symbolic address of a host
- IP address

"constructors" (call to a local DNS) :

- **`static`** `InetAddress getByName(String host)` — 1re info
- **`static`** `InetAddress[] getAllByName(String host)` — all
- **`static`** `InetAddress getLocalHost()`

- `String getHostName()`
- **`byte`**`[] getAddress()`
- `String getHostAddress()`

Remarks :

- impossibility to "create" a structure with an IP address
- `equals()` rewritten such that equality is tested with respect to the IP address

Example :

```
import java.net.*;

public class TestAdresse {
   public static void main(String[] args) {
      try {
         InetAddress localHost = InetAddress.getLocalHost();
         System.out.println("Adresse de la machine : "
                            + localHost.getHostAddress());
      }
      catch (UnknownHostException e) {}
   }
}
```

(exceptions : `UnknownHostException`, `IOException`, package `java.net`)

Non-secured TCP Connections

Client side : the class `Socket` manages the connections

- `Socket(String host, int port)`
  - opens a connection to `host` + `port`
  - if `host` does not listen on `port`
  - then return with a `IOException`
- `Socket(InetAddress host, int port)`
- `Socket(String host, int port,`
  `InetAddress interface, int portLocal)`
  - specifies a source port (by default the first free one)

- `InetAddress getInetAddress()` — IP (of the server)
- `int getPort()` — distant port
- `int getLocalPort()` — local port
- `InetAddress getLocalAddress()`

- `InputStream getInputStream()` — input stream
- `OutputStream getOutputStream()` — output stream
- `synchronized void close()` — closes the socket

Example :

```
import java.io.*; import java.net.Socket;

public class Main {
  public static void main(String[] args) {
    Socket connexion = null;
    try {
      connexion = new Socket("www.univ-paris13.fr",80);
      Writer output = new OutputStreamWriter(
      connexion.getOutputStream(), "8859_1");

      output.write("GET␣/␣HTTP␣1.0\r\n\r\n"); output.flush();
      connexion.shutdownOutput();          // partial close

      BufferedReader input =
         new BufferedReader(
           new InputStreamReader(connexion.getInputStream(),"8859_1"),
           1024);                          // input stream

      StringBuffer sb = new StringBuffer(); int c;
      while ((c = input.read()) != -1) sb.append((char) c);
         System.out.println(sb);
      } catch (IOException e) {System.out.println(e);}
      finally {
         try {if (connexion != null) connexion.close();}
         catch (IOException e) {System.out.println(e);}
}   }   }
```

Server side : the class `ServerSocket` manages the clients

- `ServerSocket(int port)` begins to listen for incoming requests on the port
  (buffer size to 50 and on all IP addresses by default)
- `ServerSocket(int port, int size)` (with buffer size specified)
- `ServerSocket(int port, int size, InetAddress adr)`
  with IP address specified `adr`
- `Socket accept()` waits for incoming requests
- `void close()` closes the server

```
import java.util.concurrent.*;
import java.net.*; import java.io.*;
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize)
       throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }

   public void run() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) { pool.shutdown(); }
   }

    public static void main(String[] args) {
      try {
       NetworkService networkService = new NetworkService(33333,5);
        networkService.run();
      } catch (IOException e) {System.out.println(e);}
    }
}
```

```
import java.util.concurrent.*;
import java.net.*; import java.io.*;
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize)
       throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }

   public void run() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) { pool.shutdown(); }
   }

    public static void main(String[] args) {
      try {
       NetworkService networkService = new NetworkService(33333,5);
        networkService.run();
      } catch (IOException e) {System.out.println(e);}
    }
}
```

Creation of the pool

Execution by a thread of the pool

```
class Handler implements Runnable {
  private final Socket socket;
  Handler(Socket socket) { this.socket = socket; }

  public void run() {
    try {
        InputStream in = socket.getInputStream();
        int i;
        while ((i = in.read()) != 0) { System.out.write(i); }
    }
        catch (SocketException e) {System.out.println(e);}
        catch (IOException e) {System.out.println(e);}
    try { socket.close(); }
        catch (IOException e) {System.out.println(e);}   }
}
```

```
class Handler implements Runnable {
  private final Socket socket;
  Handler(Socket socket) { this.socket = socket; }

  public void run() {
    try {
        InputStream in = socket.getInputStream();
        int i;
        while ((i = in.read()) != 0) { System.out.write(i); }
    }
        catch (SocketException e) {System.out.println(e);}
        catch (IOException e) {System.out.println(e);}
    try { socket.close(); }
        catch (IOException e) {System.out.println(e);}   }
}
```

Necessary as used as a thread

Be careful for reading and writing with sockets :

- When sending a string, put an explicit `\n` at the end of your string you want to send through the socket.
- Then you are able to read a line correctly, otherwise the communication is frozen.

Secured TCP Connections

Packages :

- `javax.net.ssl` abstract classes for secured communication
- `javax.net` secured sockets
- `java.security.cert` SSL keys management
- `com.sun.net.ssl` cipher algo (==> provider)

Needs to specify

- if there is authentication (e.g. RSA)
- if block encryption (e.g. DES, RC4)
- if control of the signature, data integrity (e.g. MD5, SHA)

Exhaustive method :

1. specify the provider for the cipher algo
   - either by specifying it in the file `java.security`
   - either using `addProvider(..)` from class `Security`
2. create the cipher factorywith the class `SSLSocketFactory` (by default, use `getDefault()`)
3. create the socket by means of `createSocket(...)` from class `SSLSocketFactory`

More precisely :

1. generate public and private keys and certificates (command `keytool`)
2. authenticate the certificates (==> tiers)

3. create an instance of `SSLContext` for the cipher algo
4. create an instance of `KeyManagerFactory` for the key manager
5. create an instance of `KeyStore` for specifying the file containing keys and certificates
6. initialize the `KeyManagerFactory`
7. initialize the `SSLContext`

8. create an instance of `SSLServerSocketFactory` for generating the server
9. create an instance of `SSLServerSocket`
10. create a socket with `SSLServerSocket`

cipher protocols may be managed :

- `public abstract String[] getEnabledCypherSuites()`
  returns the list of cipher protocols (one string per protocol)
- `public abstract void setEnabledCypherSuites(String[] c)`
  sets protocols (from strings `c`)

Examples :

- **SERVER Side**

```
% keytool -genkey -keystore Fichier_Certif
// generation of a file that contains the keys
// (a certificate, a private key and a public key)

% keytool -list -keystore Fichier_Certif
// if one wants to retrieve the content of the file

% keytool -selfcert -keystore Fichier_Certif
// self-certification of the public key
//       (in Fichier_Certif)

% java -Djavax.net.debug=ssl:handshake:verbose \
       ServerMaitre 20000 certif
// certif : password to "enter" the file that contains keys

// or

% java -Djavax.net.ssl.keyStore=cacerts \
       -Djavax.net.ssl.keyStorePassword=certif \
       -Djavax.net.debug=ssl:handshake:verbose \
       ServerMaitreBis 20000
```

- **CLIENT side**

```
% cp Fichier_Certif cacerts
// "client" file containing the certificates

% java -Djavax.net.ssl.trustStore=cacerts \
       -Djavax.net.debug=ssl:handshake:verbose \
       Client localhost 20000 Client.java

// -Dxxx : to specify a value in the virtual machine
```

Example of a client :

```
import java.io.*;
import java.security.*;
import javax.net.ssl.*;

public class ClientHTTPS {

   private final int portHTTPS = 443;     // default https port
   private SSLSocket sslSocket;
   private String host;

   public ClientHTTPS(String host) throws Exception {
      Security.addProvider(
           new com.sun.net.ssl.internal.ssl.Provider());
      System.setProperty("javax.net.ssl.trustStore",
                         "jssecacerts");
      SSLSocketFactory factory =
            (SSLSocketFactory) SSLSocketFactory.getDefault ();
      try {
         sslSocket = (SSLSocket) factory.createSocket(host,portHTTPS);
      } catch (IOException e) {System.out.println(e);}
   }
...// cf page suivante
```

```
...
   public static void main(String[] args) throws Exception {
      if (args.length == 0) {
          System.out.println("Usage : java ClientHTTPS host");
          return;
      }
      ClientHTTPS clientHTTPS = new ClientHTTPS(args[0]);
      clientHTTPS.test();
      clientHTTPS.close();
   }
...// cf page suivante
```

```
...
   public void test() {
     try {
      Writer output =
           new OutputStreamWriter(sslSocket.getOutputStream());
      output.write("GET␣https://"+host+"/␣HTTP␣1.1\r\n\r\n");
      output.flush();


      BufferedReader input = new BufferedReader(
      new InputStreamReader(sslSocket.getInputStream()));

      int c;
      while ((c=input.read())!=-1){System.out.write(c);}

      output.close(); input.close();

     } catch (IOException e) {System.out.println(e);}
   }

   public void close() {
      try {sslSocket.close();}
      catch (IOException e) {System.out.println(e);}
   }
}
```

Example of a SSL server :

```
...
   private SSLServerSocket serverSocket;
   private Socket socket;
   private int port;

   public ServerMaitre (int port, String password) {
      try {setPort(port);}
      catch (Exception e) { System.out.println("incorrect port"); }

// 1) creation of the factory for the cipher algo
      KeyManagerFactory kmf=null;
      SSLContext context=null;
      try {
         context = SSLContext.getInstance("TLS");
         kmf = KeyManagerFactory.getInstance("SunX509");
      } catch (NoSuchAlgorithmException e1) { e1.printStackTrace(); }

// 2) specification of the key manager
      KeyStore ks = null;
      try {
         ks = KeyStore.getInstance("JKS");
      } catch (KeyStoreException e2) { e2.printStackTrace(); }
... // cf page suivante
```

```
...
// 3) retrieval of the cerificate and the key
      char[] passPhrase = password.toCharArray();
      try {
         ks.load(new FileInputStream("Fichier_Certif"), passPhrase);
      } catch (NoSuchAlgorithmException e3) { e3.printStackTrace();
      } catch (CertificateException e3) { e3.printStackTrace();
      } catch (FileNotFoundException e3) { e3.printStackTrace();
      } catch (IOException e3) { e3.printStackTrace();
      }

// 4) initialization of the key manager (certificate, key, password)
      try {
         kmf.init(ks, passPhrase);
      } catch (KeyStoreException e4) { e4.printStackTrace();
      } catch (NoSuchAlgorithmException e4) { e4.printStackTrace();
      } catch (UnrecoverableKeyException e4) { e4.printStackTrace();
      }
... // cf page suivante
```

```
...
// 5) specification of the context for generating a SSLServerSocket
      try {
         context.init(kmf.getKeyManagers(), null, null);
      } catch (KeyManagementException e5) { e5.printStackTrace();
      }

// 6) creation of the factory for SSLServerSocket
      SSLServerSocketFactory factory =
            context.getServerSocketFactory ();

// 7) creation of an instance of SSLServerSocket
      try {
         serverSocket =
            (SSLServerSocket) factory.createServerSocket (this.port);
         System.out.println ("Création␣Socket␣OK");
      }
      catch (IOException e) {
         System.out.println ("Erreur␣ServerSocket␣:␣" + e);
         System.exit (0);
      }
   }
...
```

A shorter form !

```
...
   private SSLServerSocket serverSocket;
   private Socket socket;
   private int port;

   public ServerMaitreLight (int port) {
     try {setPort(port);}
     catch (Exception e) { System.out.println("incorrect port"); }

      SSLServerSocketFactory sslSSF =
        (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();

      try {
      serverSocket =
          (SSLServerSocket) sslSSF.createServerSocket(port);
        System.out.println("SSL ServerSocket started");
     }
     catch (IOException e) {
      System.out.println ("ServerSocket : " + e);
      System.exit (0);
     }
   }
...
```

There is also the possibility to define specific rights for connections to a server :

- `Permission p = new java.net.SocketPermission ("F205-2.ig-edu.univ-paris13.fr", "connect");`
- `Permission p = new java.net.SocketPermission ("*.ig-ens.univ-paris13.fr:1000-3000","accept");`

Two methods :

- a file `java.policy` specifying the rights (configuration file `$JDKHOME/jre/lib/security/java.policy` loaded when launching the Java VM)
- a class redefining the security policy

UDP Connections

- advantage : quick
- default : unsafe
- to be used only when connections consider small packets
- data managed by the class `DatagramPacket` with methods `receive()` and `send()`

Example :

```
import java.net.*;   import java.io.*;

public abstract class ServerUDP extends Thread {
   private int sizeBuffer; protected DatagramSocket ds;
   public ServerUDP(int port, int sizeBuffer) throws SocketException {
      this.sizeBuffer = sizeBuffer;
      this.ds = new DatagramSocket(port);
   }

   public ServerUDP(int port) throws SocketException {
      this(port, 8192);
   }

   public void run() {
      byte[] buffer = new byte[sizeBuffer];
      while (true) {
         DatagramPacket input =
               new DatagramPacket(buffer, buffer.length);
         try { ds.receive(input); this.manage(input);
         } catch (IOException e) {}
   }   }

   public abstract void manage(DatagramPacket packet);
}
```

```
import java.net.*;   import java.io.*;

public class EchoServerUDP extends ServerUDP {
   public final static int PORT = 5007;   // echo = 7
   public EchoServerUDP() throws SocketException {super(PORT);}

   public void manage(DatagramPacket packet) {
      try {
         System.out.println(new String(packet.getData()));
         DatagramPacket output = new DatagramPacket(
            packet.getData(),
            packet.getLength(),
            packet.getAddress(),
            packet.getPort());
         ds.send(output);
      } catch (IOException e) {}
   }

   public static void main(String[] args) {
      try {
         EchoServerUDP server = new EchoServerUDP();
         server.start();
      } catch (SocketException e) {}
}   }
```

Example to receive packets :

```
import java.net.*; import java.io.*;

public class ReceiverThread extends Thread {
   private DatagramSocket socket;
   private boolean quit = false;

   public ReceiverThread(DatagramSocket ds) throws SocketException
                        { this.socket = ds; }
   public void halt() { this.quit = true; }

   public void run() {
      byte[] buffer = new byte[65507];

      while (true) {
         if (quit) return;
         DatagramPacket input =
            new DatagramPacket(buffer, buffer.length);

         try {
            socket.receive(input);
            String s = new String(input.getData(), 0,
                                  input.getLength());
            System.out.println(s);
            Thread.yield();
         } catch (IOException e) {}
}    }    }
```

Multicast Connections

In case of shread communications with user groupes (video, newsgroups, ...), Multipoint connections is a good solution :

- a datagram is shared worldwide. Internet routersLes distribute packets (without duplication) wrt a distance.
- Each user groups is referenced by a specific IP address.
- UDP protocol is at use
- packets are sent onto a "geographic zone"

Mechanisms are fundamentally similar to UDP datagrams. End users have to join (or quit) a group :

- `MulticastSocket(int port)` creates a multipoint socket
- `MulticastSocket joinGroup (InetAddress ia)` method allowing to join a group referenced by an IP address `ia`
- `MulticastSocket leaveGroup (InetAddress ia)` method to leave a group
- `MulticastSocket receive(DatagramPacket dp)` to receive a UDP datagram
- `MulticastSocket send(DatagramPacket dp, byte ttl)` to send a UDP datagram (ttl specifies the number of routers the datagram may go through)

Classes `URL` and `URLConnection`

- `URL` : management of the object URL (i.e. static information linked to an URL)
- `URLConnection` : management of the connections to an URL (i.e. management of the socket)

(exception : `MalformedURLException`)

```
public URL(String url)
    // new URL("http://www.univ-paris13.fr/index.html");
    // exception if protocol unknown by the VM
public URL( String protocole,     // http
            String machine,       // www.univ-paris13.fr
            String fichier)       // index.html

public URL( String protocole,
          String machine,
          int port,           // 80 (by default for http)
          String fichier)

public URL( URL base, String relative)

public URL( String protocole,
          String machine,
          int port,
          String fichier,
          URLStreamHandler handler)
    // to specify the protocol manager
```

```
String   getProtocol(), getHost(), getFile(), getPath(),
         getRef(), getQuery(), getUserInfo(), getAuthority()

int      getPort()
```

Input/Output :

`URLConnection openConnection()`

opens a connection to an URL,
returns a socket to this URL
(hence input/output streams)

`InputStream openStream()`

opens a connection to an URL,
authentication if necessary,
instantiates an `InputStream` to fetch data

`Object getContent()`

fetches data from an URL,
then formats data as an instance of Object
the format is specified wrt the protocol.
`getClass()` on this object should allow to know which kind of object it is
(e.g. `URLImageSource` for a gif object, `MeteredStream` for an applet, ...)

```
(URL) getContent() == openConnection().getContent()
(URL) openStream() == openConnection().getInputStream()
```

The connection to an URL being opened, one can retrieve/set various information :

- fetch the specification of this URL
- configure the connection
- send/receive data

Specifications of an URL :

```
String getContentType()
```
type MIME (text/html, image/gif)

```
int getContentLength()
String getContentEncoding()
```
(null if no encoding, otherwise x-gzip, ...)

```
long getDate()
getExpiration()
getLastModified()
getHeaderField(String header)
```
("content-type", ...)

Configuration of the connection for

- managing the cache,
- setting a password,
- specifying ythe header when sending requests

Input/Output of data

- `InputStream getInputStream()`
- `OutputStream getOutputStream()`

Existence of specific sub-classes :

- http : `HttpURLConnection`, `setRequestMethod()`, ...
- jar : `JarURLConnection`, `getJarEntry()`, ...
- ...

Example :

```
import java.net.*; import java.io.*;

public class Mailer {

  public static void main(String[] args) {
    System.setProperty("mail.host", "smtp.orange.fr");
    try {
      URL urlMail = new URL("mailto:cf@lipn.univ-paris13.fr");
      URLConnection con = urlMail.openConnection();
      PrintStream p = new PrintStream(con.getOutputStream());
      p.println("Subject: test\r\n\r\n corps du mail");
         // a message is defined as a header
         // followed by an empty line, then message body
      p.close();
   } catch (IOException e) {System.out.println(e);}
} }
```

`%java Mailer`

or

`%java -Dmail.host=smtp.orange.fr Mailer`

if `System.setProperty` not in the program, where `mail.host` is a property needed for specifying the address of the SMTP server.

Management of protocols

It is possible to define its own protocols (either because insufficiently predefined or because one wants to provide a new protocol)

Let us look at the implementation of the class `URL` :

```
public final class URL implements java.io.Serializable {

  private String protocol;
  private String host;
  ...
  transient URLStreamHandler handler;
      // transient = pas de sérialisation
  public URL(String protocol, String host, int port,
             String file, URLStreamHandler handler) {
      this.host = host;
      this.port = port;
      ...
      this.handler = getURLStreamHandler(protocol)
  }
  public URLConnection openConnection() throws java.io.IOException {
      return handler.openConnection(this);
  }
... // following page
```

```
static URLStreamHandlerFactory factory;

static URLStreamHandler getURLStreamHandler(String protocol){

   // Use the factory (if any)
   if (factory != null)
      handler = factory.createURLStreamHandler(protocol);
   // Try java protocol handler
   if (handler == null) {
   ...
   packagePrefixList += "sun.net.www.protocol";
   ...
   try {
      String clsName = packagePrefix+"."+protocol +".Handler";
      Class cls = null;
      try { cls = Class.forName(clsName);
      } catch (ClassNotFoundException e) {
         ClassLoader cl = ClassLoader.getSystemClassLoader();
         if (cl != null) { cls = cl.loadClass(
      }
      if (cls != null) {
         handler  = (URLStreamHandler)cls.newInstance();
      }
   } catch (Exception e) { ... }

   return handler;
}
```

Full example (4 pages) :

```
import java.net.*; import java.io.*;

public class GetGridApp {

 public static void main(String args[]){
  try{
   GridFactory gridFactory = new GridFactory();
   URLConnection.setContentHandlerFactory(gridFactory);
   if(args.length!=1) error("Usage: java GetGridApp URL");
   URL url = new URL(args[0]);
   CharGrid cg = (CharGrid) url.getContent();
   for(int i=0;i<cg.height;++i) {
    for(int j=0;j<cg.width;++j) {
     if(cg.values[i][j]) System.out.print(cg.ch);
     else System.out.print(" ");
    }
    System.out.println();
   }
  }catch (MalformedURLException ex){ error("Bad URL");
  }catch (IOException ex){ error("IOException occurred."); }
 }

 public static void error(String s){
  System.out.println(s); System.exit(1);
 }
}
```

```
import java.net.*; import java.io.*;

class GridFactory implements ContentHandlerFactory {

 public GridFactory() { }

 public ContentHandler createContentHandler(String mimeType) {
  if(mimeType.equals("text/cg")) {
   System.out.println("Requested␣mime␣type:␣"+mimeType);
   return new GridContentHandler();
  }
  return new GridContentHandler();
 }

}
```

```
public class CharGrid {
 public int height;
 public int width;
 public char ch;
 public boolean values[][];

 public CharGrid(int h,int w,char c,boolean vals[][]) {
  height = h; width = w; ch = c; values = vals; }

}
```

```
import java.net.*;
import java.io.*;

public class GridContentHandler extends ContentHandler {

 public Object getContent(URLConnection urlc)
         throws IOException {
  DataInputStream in = new DataInputStream(urlc.getInputStream());
  int height = (int) in.readByte() - 48;
  int width = (int) in.readByte() - 48;
  char ch = (char) in.readByte();
  boolean values[][] = new boolean[height][width];

  for(int i=0;i<height;++i) {
   for(int j=0;j<width;++j) {
    byte b = in.readByte();
    if(b == 48) values[i][j] = false;
    else values[i][j] = true;
   }
  }

  in.close();
  return new CharGrid(height,width,ch,values);
 }
}
```

Fichier charGrid.cg :

```
55O1000101010001000101010001
```

Exécution :

```
% java GetGridApp file://localhost/home/cf/.../charGrid.cg
```

3 Network : adressing and sockets

- Networking
- URL
- RMI and RMI-IIOP

RMI :

- Mechanism developed by Sun for distributed systems, i.e. the whole "program" is distributed on different virtual machines
- a method implemented in a VM A may be called by a VM B "as if" this method were on B
- Mechanism specific to Java
- Uses the library `java.rmi`

RMI-IIOP :

- IIOP : *Internet Inter-Orb Protocol*
- Compatibility with CORBA (hence usable with other environments/languages)
- Necessary with the environnement EJB (*Enterprise Java Beans*)
- Uses the library `java.rmi` and `javax.rmi`

Principales différences :

- Distinct classes for identifying distant objects :
  - RMI : `java.rmi.server.RemoteObject`
  - RMI-IIOP : `javax.rmi.PortableRemoteObject`

3 virtual machines :

- A : contains objects (i.e. instances, codes) other virtual machines may use
- B : virtual machine calling methods of objects in A "as if" these objects were in B
- C : virtual machine presenting objects of A for use to other virtual machines, say B here (similar to a DNS)

Principle :

- Launching of the server C : server waiting for requests on port 1099 (by default) : release of objects or retrieval of information about objects (the command `rmiregistry` launches such a server)
- Launching of server A :
  - creation of objects in the VM A that could be declared on C
  - opening of a port waiting for requests about published objects
  - request to C to publish the objects (by means of information about A socket : port, IP address, symbolic name for the object)
- Launching of server B :
  - request to C to fetch information about on object using the symbolic name
  - execution of methods of this object

To be more complete : requires specific classes for serializing and deserializing requests and datas (arguments for methods).

- B's codes should contain an interface to specify the type of the distant object.
- Codes for serializing and deserializing are dynamically generated by A, and sent to B before the exchange of requests and data can be done (*stub* and *skeleton*)
- Before version 5, these codes were statically, say manually, generated by the command `rmic`

Example (step 1) :

Interface to be present on A and B : class `Display.java`

```
import java.rmi.*;
import java.io.*;

public interface Display extends Remote{
   public void showDigest(File fileName) throws RemoteException;
}
```

Example (step 2) :

Implementation of the interface on A : class `DisplayClass.java`

```
import java.rmi.*;
import java.io.*; import java.security.*;

public class DisplayClass implements Display{
   File input; byte[] digest;

   public void showDigest(File input) throws RemoteException{
      try {
        this.input = input;
        MessageDigest sha = MessageDigest.getInstance("SHA");
        DigestInputStream din = new DigestInputStream(new FileInputStream(input),sha);

        while ((din.read()) != -1); din.close();
        digest = sha.digest();
        System.out.println(this);
      }
      catch(IOException e) {System.err.println(e);}
      catch(NoSuchAlgorithmException  e) {System.err.println(e);}
   }

   public String toString() {
      String result = input.getName() + ":␣";
      if (digest != null) {
         for (int i = 0; i< digest.length; i++) {result += digest[i] + "␣";}
      } else { result += "unusable␣digest";};
      return result;
   }
}
```

Example (step 3) :

Compilation of the interface and its implementation :

```
% javac Display.java       // ==> Display.class
% javac DisplayClass.java  // ==> DisplayClass.class
```

Example (step 4) :

A code `DisplayPublisher.java` to create an object and publish it via the RMI server C

```
import java.rmi.*;
import java.rmi.server.*;

public class DisplayPublisher {
   public static void main (String[] args) throws Exception {
      Display display = new DisplayClass();
      UnicastRemoteObject.exportObject(display);
         // creates a thread waiting for calls on this object (here display),

      Naming.rebind("/UnDisplayDistant", display);
         // the class Naming publishes the object as a symbolic name on C
         // URL :: rmi://host:port/nom
         // bind : the name should not exist
         // rebind : using or reusing of the name
         // unbind : freeing of the link
   }
}
```

Example (step 5) :
Publish the object on a RMI server

```
% rmiregistry &
      // Creation of a name server by default on port 1099
% javac DisplayPublisher.java
% java DisplayPublisher
```

The last command *creates* the virtual machine A that stays running.

Example (step 6) :

Code `DisplayClient.java` to be run on B

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;

public class DisplayClient {

   public static void main(String[] args) throws Exception{
      File file = new File(args[0]);

      Display display = (Display) Naming.lookup("rmi://localhost/UnDisplayDistant");

      display.showDigest(file);
   }

}
```

Remarks :

- Several distant calls may be done on the same object : each such call is executed by a particular thread. Concurrency is then to be carefully considered.
- Parameters of a call are sent to the server : parameters with non-primitive types cannot be modified.

RMI uses implicitly dynamic loading of classes, that may be used as such outside RMI :

- `java.lang.ClassLoader` : generic abstract class
- `getSystemClassLoader()` : standard loader

Example :

```
public interface TestInterface {
   public int somme(int x, int y);
}
```

```
public class Test implements TestInterface {
   public int somme(int x, int y){ return x+y; }
}
```

```
public class TestLoadClass {
   public static void main(String[] args) throws Exception{
      ClassLoader loader = ClassLoader.getSystemClassLoader();

//    Object main = loader.loadClass("Test").newInstance();
//    System.out.println(main.getClass());

      TestInterface main = (TestInterface) loader.loadClass("Test").newInstance();
      System.out.println(main.somme(3,4));
   }
}
```

Sub-classes with specific loader :

- `security.SecureClassLoader` : secure loader
- `java.net.URLClassLoader` : loader by means of URL addresses
- `rmi.server.RMIClassLoader` : specific to RMI

Example : (we suppose that `Test.class` is in the folder `/home/cf/TMP/URL/Bibli/`)

```
import java.net.*;
import java.util.*;

public class TestLoadClass {

   public static void main(String[] args) throws Exception{

   URL url = new URL("file:///home/cf/TMP/URL/Bibli/");
   URLClassLoader loader = new URLClassLoader(new URL[] {url});

   TestInterface main = (TestInterface) loader.loadClass("Test").newInstance();
   System.out.println(main.somme(3,4));

   }
}
```

4 Annotations management

An **annotation** is a *meta-data* for allowing the “environnement”

- to produce various information (documentation, tests, logs, ...)
- to generate configuration files (for the deployment of a final system)
- to generate interfaces (to precise methods to be included, to serve in a RMI system), subsidiary classes, ...
- to specify constants (e.g. information concerning a data base to be used)

- An annotation begins with the symbol `@`.
- L'environnement doit comprendre des programmes permettant d'interpréter les annotations.
- An annotation is interpreted either by the compiler or by the Java interpreter (when a meta-annotation `@Retention` is used when defining the annotation).
- Annotations are available since version 5.

- An annotation may be used as a modifier before any modifier (e.g. **`public`**) of a class, of a method.
- It may be combined in a sequence of annotations
- An annotation may also annotate another annotation

```
...
@MonAnnotation(
   unAttribut = 12345,
   unAutreAttribut = "une valeur",
)
public static void maMethode(...) { ... }
...
```

ou

```
...
@UneAutreAnnotation
public class MaClasse(...)
   { ... }
...
```

ou

```
...
@UneDerniereAnnotation("une valeur")
public class MaClasse(...)
   { ... }
...
```

(if the annotation has a unique attribute, say field, it is by default `String value()`)

- An annotation is *defined* as a kind of interface, where methods are replaced by attributes :

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
public @interface MonAnnotation {
   int    unAttribut();
   String unAutreAttribut();
}
```

- An annotation is *used* in a code :

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

public class CodeMonAnnotation {
   public static void main(String[] args) throws Exception {
      for (Method m : Class.forName(args[0]).getMethods()) {
         if (m.isAnnotationPresent(MonAnnotation.class)) {
         ... // code à effectuer
         }
      }
   }
}
```

For the previous example, the annotation has to be kept in the file `.class`, hence the need for the annotation `Retention`

For each annotation, `javac` seeks in its environment (say `classpath`) a class with the name of the annotation.
More than 60 annotations are standard :

- `@Deprecated` : before a method, indicates that this method is not recommended (i.e. its use will generate a warning)
- `@Override` : before a method, indicates the method should overload a definition already given in a super-class
- `@SuppressWarnings(type)` : removes warnings for the type (`"deprecation"`, `"all"`,...)

Example : standard annotation in a Java code

```
import java.io.*;
public class Test {
 // @SuppressWarnings("fallthrough")
 public static void main(String args[]) {
   PrintStream out = System.out;
   int i = args.length;
   switch (i) { // manque des breaks
     case 0:  println("0");
     default: println("Default");
   }
 }
 // @Override
 public String toZtring () {
   return super.toString();
 }
}
```

Without removing comments :

```
$ javac Test.java
$ javac -Xlint Test.java
Test.java:7: warning: [fallthrough]
     possible fall-through into case
   default: System.out.println("Default");
   ^
1 warning
$
```

With `@Override` uncommented :

```
$ javac Test.java
Test.java:10: method does not override
  or implement a method from a supertype
   @Override
   ^
1 error
$
```

With `@SuppressWarnings` uncommented :

```
$ javac -Xlint Test.java
$
```

Meta-annotations are useful for defining annotation characteristics. For example :

- `@Retention(type)` : meta-annotation that keeps wrt the type (`RetentionPolicy.RUNTIME` in the executable, `RetentionPolicy.CODE` not in the executable) the annotation that follows.
- `@Target(type)` : meta-annotation that precises to which kind of elements the annotation may be applied (`ElementType.METHOD` for a method, `ElementType.TYPE` for a class, an interface, ...).

Example : Manual management during the execution (1)

```java
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Audit {
    boolean value() default false;
}
```

```java
public class Test {
   public static void main(String args[]) {
      Application app = new Application();
      app.methA("a1");
      app.methB();
   }
}
```

```java
public class Application {
  @Audit(true)
  public void methA(String s) {
    GestionAudit.gestion(this,"methA",String.class);
    // code de l'application
  }

  public void methB() {
    GestionAudit.gestion(this,"methB");
    // code de l'application
  }
}
```

```
$javac *.java
$java Test
[Audit] appel de methA
$
```

Example : Manual management during the execution (1)

```java
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Audit {
    boolean value() default false;
}
```

```java
public class Test {
   public static void main(String args[]) {
      Application app = new Application();
      app.methA("a1");
      app.methB();
   }
}
```

the annotation Audit will be kept for use during the execution

```java
public class Application {
  @Audit(true)
  public void methA(String s) {
    GestionAudit.gestion(this,"methA",String.class);
    // code de l'application
  }

  public void methB() {
    GestionAudit.gestion(this,"methB");
    // code de l'application
  }
}
```

```
$javac *.java
$java Test
[Audit] appel de methA
$
```

Example : Manual management during the execution (1)

```java
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Audit {
    boolean value() default false;
}
```

statement of the annotation

```java
public class Test {
   public static void main(String args[]) {
      Application app = new Application();
      app.methA("a1");
      app.methB();
   }
}
```

```java
public class Application {
  @Audit(true)
  public void methA(String s) {
    GestionAudit.gestion(this,"methA",String.class);
    // code de l'application
  }

  public void methB() {
    GestionAudit.gestion(this,"methB");
    // code de l'application
  }
}
```

call during the execution to a class that will manage the annotation (cf following slide)

```
$javac *.java
$java Test
[Audit] appel de methA
$
```

Example : Manual management during execution (2)

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

public class GestionAudit {
   public static void gestion(Object object, String methodName, Class... paramTypes) {
      try {
         Class paramTypesArray[] = new Class[paramTypes.length];
         int i = 0;
         for (Class paramType : paramTypes) paramTypesArray[i++] = paramType;

   // To recover the method that was called
         Method method = object.getClass().getMethod(methodName, paramTypesArray);

   // If not annotated, resume the execution of the program
         if( !method.isAnnotationPresent(Audit.class) ) return;

   // To recover the data associated to the annotation, then execute what has to be done
         Audit auditValue = method.getAnnotation(Audit.class);
         if( auditValue.value() ) audit("[Audit] appel de " + methodName);
      } catch (Exception e) { audit("[Audit exception] sur l'appel de " + methodName);
      }
   }
   private static void audit(String msg) { System.err.println(msg); }
}
```

Example : Manual management during execution (2)

variable number of arguments

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

public class GestionAudit {
   public static void gestion(Object object, String methodName, Class... paramTypes) {
      try {
         Class paramTypesArray[] = new Class[paramTypes.length];
         int i = 0;
         for (Class paramType : paramTypes) paramTypesArray[i++] = paramType;

   // To recover the method that was called
         Method method = object.getClass().getMethod(methodName, paramTypesArray);

   // If not annotated, resume the execution of the program
         if( !method.isAnnotationPresent(Audit.class) ) return;

   // To recover the data associated to the annotation, then execute what has to be done
         Audit auditValue = method.getAnnotation(Audit.class);
         if( auditValue.value() ) audit("[Audit] appel de " + methodName);
      } catch (Exception e) { audit("[Audit exception] sur l'appel de " + methodName);
      }
   }
   private static void audit(String msg) { System.err.println(msg); }
}
```

Java reflexivity

Annotations are often managed by the compiler :

- In version 5 : the tool `apt` (*annotation processing tool*) processes annotations.
- Since version 6 : `javac` includes the processing of annotations while computing :
  - a structure corresponding to the source code
  - a structure corresponding to the abstract syntax tree of the code

- Hence since version 6, two ways to look at the code :
  - (`javax.lang.model.element`) the interface `Element` and its sub-interfaces characterize a block of a code (package, type, class, method). Hence it allows to manipulate the source code.
  - (e.g. `com.sun.source.tree`) the interface `Tree` and its sub-interfaces help to know the abstract syntax tree as it is generated by the compiler. It allows also to control the source file.
  - In the 2 cases, the pattern *visitor* is used (method `accept(v)` where `v` is a visitor, a kind of iterator on the structure (either `ElementVisitor` or `TreeVisitor`) that implements methods `visitXXX()` (see next slide)

Example : Management during the compilation (1)

```
public class Test {
   public static void main(String args[]) {
      Application app = new Application();
      app.methA("a1");
      app.methB("b");
      app.methC();
      app.methA("a2");
   }
}
```

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Audit {
    // if true, calls to this method are logged
    boolean value() default false;
}
```

```
public class Application {
    @Audit(true)
    public void methA(String s) {
      int i = 0;
      // code of the method
    }

    @Audit(false)
    public void methB(String s) {
      // code of the method
    }

    @Deprecated
    public void methC() {
      // code of the method
    }
}
```

Example : Management during the compilation (2)

```
%javac Application.java
%javac Test.java
Note: Test.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

```
%javac -cp $CLASSPATH:/opt/jdk1.6.0_10/lib/tools.jar GestionAudit.java
%javac -processor GestionAudit Application.java
methA(java.lang.String) :
Valeur de l'annotation associee = true
METHOD
Method methA: null
Body: {
    int i = 0;
}
methB(java.lang.String) :
Valeur de l'annotation associee = false
METHOD
Method methB: null
Body: {
}
```

annotations Audit and Deprecated kept in the code

Example : Management during the compilation (2)

```
%javac Application.java
%javac Test.java
Note: Test.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

the code annotated by Deprecated generates a warning

```
%javac -cp $CLASSPATH:/opt/jdk1.6.0_10/lib/tools.jar GestionAudit.java
%javac -processor GestionAudit Application.java
methA(java.lang.String) :
Valeur de l'annotation associee = true
METHOD
Method methA: null
Body: {
    int i = 0;
}
methB(java.lang.String) :
Valeur de l'annotation associee = false
METHOD
Method methB: null
Body: {
}
```

Example : Management during the compilation (2)

```
%javac Application.java
%javac Test.java
Note: Test.java uses or overrides a depreca
Note: Recompile with -Xlint:deprecation for
```

jar that contains classes to manipulate the abstract tree structure

```
%javac -cp $CLASSPATH:/opt/jdk1.6.0_10/lib/tools.jar GestionAudit.java
%javac -processor GestionAudit Application.java
methA(java.lang.String) :
Valeur de l'annotation associee = true
METHOD
Method methA: null
Body: {
    int i = 0;
}
methB(java.lang.String) :
Valeur de l'annotation associee = false
METHOD
Method methB: null
Body: {
}
```

the code annotated by Audit is managed

Example : Management during the compilation (3)

```
import java.util.*;
import javax.annotation.processing.*;
import javax.lang.model.*;
import javax.lang.model.element.*;

@SupportedAnnotationTypes(value= {"Audit"})
@SupportedSourceVersion(SourceVersion.RELEASE_6)
public class GestionAudit extends AbstractProcessor {
   private ProcessingEnvironment environment;

   @Override public void init(ProcessingEnvironment environment) {
      super.init(environment);
      this.environment = environment;
   }

   @Override public boolean process(Set<? extends TypeElement> annotations,
                           RoundEnvironment roundEnv){
      AuditVisitor auditVisitor=new AuditVisitor(environment);
      for (Element element : roundEnv.getElementsAnnotatedWith(Audit.class)){
          System.out.println(element + " : ");
          System.out.println("Valeur de l'annotation associée = "
                             + element.getAnnotation(Audit.class).value());
          System.out.println(element.getKind());
          auditVisitor.visit(element,null);
      }
      return false;
   }
}
```

Example : Management during the compilation (3)

```
import java.util.*;
import javax.annotation.
import javax.lang.model.
import javax.lang.model.element.*;

@SupportedAnnotationTypes(value= {"Audit"})
@SupportedSourceVersion(SourceVersion.RELEASE_6)
public class GestionAudit extends AbstractProcessor {
   private ProcessingEnvironment environment;

   @Ove                                    ocessingEnvironment environment) {
      s
      this.environment = environment;
   }

   @Override public boolean process(Set<? extends TypeElement> annotations,
                           RoundEnvironment roundEnv){
      AuditVisitor auditVisitor=new AuditVisitor(environment);
      for (Element element : roundEnv.getElementsAnnotatedWith(Audit.class)){
          System.out.println(element + " : ");
          System.out.println("Valeur de l'annotation associée = "
                             + element.getAnnotation(Audit.class).value());
          System.out.println(element.getKind());
          auditVisitor.visit(element,null);
      }
      return false;
   }
}
```

List of the annotations this class can process

The Java release should be mentioned

Example : Management during the compilation (3)

```
import java.util.*;
import javax.annotation.processing.*;
import javax.lang.model.*;
import javax.lang.model.element.*;

@SupportedAnnotationTypes(value= {"Audit"})
@SupportedSourceVersion(SourceVersion.RELEASE_6)
public class GestionAudit extends AbstractProcessor {
   private ProcessingEnvironment environment;

   @Override public void init(ProcessingEnvironment environment) {
      super.init(environment);
      this.environment = environment;
   }

   @Override public boolean process(Set<? extends TypeElement> annotations,
                           RoundEnvironment roundEnv){
      AuditVisitor auditVisitor=new AuditVisitor(environment);
      for (Element element : roundEnv.getElementsAnnotatedWith(Audit.class)){
          System.out.println(element + " : ");
          System.out.println("Valeur de l'annotation associée = "
                             + element.getAnnotation(Audit.class).value());
          System.out.println(element.getKind());
          auditVisitor.visi
      }
      return false;
   }
}
```

default processing class

instanciation (e.g. with javac) with the compilation environment

process on the current environnement

Example : Management during the compilation (4)

```
import javax.lang.model.*;                  import javax.lang.model.util.*;
import javax.lang.model.element.*;          import javax.annotation.processing.*;
import com.sun.source.tree.MethodTree;      import com.sun.source.util.Trees;

public class AuditVisitor implements ElementVisitor<Void, Void> {
   private ProcessingEnvironment environment;
   private Trees trees;

   public AuditVisitor(ProcessingEnvironment environment) {
      this.environment=environment;
      this.trees = Trees.instance(environment);
   }
   @Override public Void visit(Element e) { return visit(e,null); }
   @Override public Void visit(Element e,Void p) {
      switch(e.getKind()) {
         case METHOD:   visitMethod(e,trees); break;
         default:       visitUnknown(e, p);
      }
      return null;
   }
   private void visitMethod(Element methodElement,Trees p) {
      System.out.println("Method␣"+methodElement.getSimpleName()+":␣"
         +environment.getElementUtils().getDocComment(methodElement));
      MethodTree methodTree = (MethodTree) p.getTree(methodElement);
      System.out.println("Body:␣"+methodTree.getBody());
   }
   @Override public Void visitUnknown(Element element,Void p) {return null;}
...
}
```

Example : Management during the compilation (4)

Class to travel through the AST

Return type for the method visit

Type of the argument of the method visit

```
import javax.la                    import java
import javax.lang.model.element.*;  import javax.annotation.processing.*;
import com.sun.source.tree.MethodTree; import com.sun.source.util.Trees;

public class AuditVisitor implements ElementVisitor<Void, Void> {
   private ProcessingEnvironment environment;
   private Trees trees;

   public AuditVisitor(ProcessingEnvironment environment
      this.environment=environment;
      this.trees = Trees.instance(environment);
   }
   @Override public Void visit(Element e) { return visit(e,null); }
   @Override public Void visit(Element e,Void p) {
      switch(e.getKind()) {
         case METHOD:   visitMethod(e,trees); break;
         default:       visitUnknown(e, p);
      }
      return null;
   }
   private void visitMethod(Element methodElement,Trees p) {
      System.out.println("Method␣"+methodElement.getSimpleName()+":␣"
         +environment.getElementUtils().getDocComment(methodElement));
      MethodTree methodTree = (MethodTree) p.getTree(methodElement);
      System.out.println("Body:␣"+methodTree.getBody());
   }
   @Override public Void visitUnknown(Element element,Void p) {return null;}
...
}
```

Example : Management during the compilation (4)

```
import javax.lang.model.*;                  import javax.lang.model.util.*;
import javax.lang.model.element.*;          import javax.annotation.processing.*;
import com.sun.source.tree.MethodTree;      import com.sun.source.util.Trees;

public class AuditVisitor implements ElementVisitor<Void, Void> {
   private ProcessingEnvironment environment;
   private Trees trees;

   public AuditVisitor(P                    ironment) {
      this.environment=
      this.trees = Trees.instance(environment);
   }
   @Override public Void visit(Element e) { return visit(e,null); }
   @Override public Void visit(Element e,Void p) {
      switch(e.getKind()) {
         case METHOD:   visitMethod(e,trees); break;
         default:       visitUnknown(e, p);
      }
      return null;
   }
   private void visitMethod(Element methodElement,Trees p) {
      System.out.println("Method␣"+methodElement.getSimpleName()+":␣"
         +environment.getElementUtils().getDocComment(methodElement));
      MethodTree methodTree = (MethodTree) p.getTree(methodElement);
      System.out.println("Body:␣"+methodTree.getBody());
   }
   @Ove                                  nknown(Element element,Void p) {return null;}
...
}
```

standard method for visiting an element

Object-node of the AST (here a method). It allows for knowing its body, name, parameters, ...

An example in J2EE / link between a Java code and a data base

```
...
@Entity
@Table(name = "COMMANDE")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Commande.findAll", query = "SELECT c FROM Commande c"),
    @NamedQuery(name = "Commande.findByCommandeId", query = "SELECT c FROM Commande c
        WHERE c.commandeId = :commandeId")})
public class Commande implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "COMMANDE_ID")
    private Integer commandeId;
    @Column(name = "QUANTITE")
    private Integer quantite;
    @Column(name = "TOTAL")
    private Integer total;
    @Size(max = 20)
    @Column(name = "EMAIL")
    private String email;
    @JoinColumn(name = "PIZZA_ID", referencedColumnName = "PIZZA_ID")
    @ManyToOne
    private Pizza pizzaId;

    public Commande() {}

    public Commande(Integer commandeId) {
        this.commandeId = commandeId;
    }

// getters and setters
...}
```

Another example in J2EE / webservice

```
...
@WebService(serviceName = "WebServicePizza")
@Stateless()
public class WebServicePizza {
    @EJB
    private PizzaFacadeLocal ejbRef;

    @WebMethod(operationName = "create")
    @Oneway
    public void create(@WebParam(name = "pizza") Pizza pizza) {
        ejbRef.create(pizza);
    }

    @WebMethod(operationName = "edit")
    @Oneway
    public void edit(@WebParam(name = "pizza") Pizza pizza) {
        ejbRef.edit(pizza);
    }

    @WebMethod(operationName = "find")
    public Pizza find(@WebParam(name = "id") Object id) {
        return ejbRef.find(id);
    }

    @WebMethod(operationName = "findAll")
    public List<Pizza> findAll() {
        return ejbRef.findAll();
    }
...
}
```

A last example : JAXB for processing XML data

- JAXB is an API for reading/writing processing of XML data
- the command `xjc` generates Java classes from XML schemas. These classes contain annotations to link Java structures to XML structures.
- the class `JAXBContext` (in `javax.xml.bind`) contains methods for reading/writing XML data :
  - analysis (*unmarshall*) of XML documents and generation of a Java object (similar to the DOM mechanism)
  - verification of a Java object wrt a XML schema
  - génération (*marshall*) of a XML file from a Java object

Example : XML schema for courses (1) formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="course" type="typeCourse"/>
  <xsd:complexType name="typeCourse">
   <xsd:attribute name="title" type="xsd:string"  use="required"/>
   <xsd:sequence>
    <xsd:element ref="supervisor" />  <xsd:element ref="student"/>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="student" type="typeStudent"/>
  <xsd:complexType name="typeStudent">
    <xsd:attribute name="prenom"  type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="supervisor" type="typeResponsable"/>
  <xsd:complexType name="typeSupervisor">
   <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
 </xsd:schema>
```

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<course intitule="M2PLS">
 <supervisor nom="Boudes"/>
 <student prenom="Jean" nom="Dupond"/><student prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Example : XML schema for courses (1) formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="course" type="typeCourse"/>
  <xsd:complexType name="typeCourse">
   <xsd:attribute name="title" type="xsd:string"  use="required"/>
   <xsd:sequence>
    <xsd:element ref="supervisor" />  <xsd:element ref="student"/>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="student" type="typeStudent"/>
  <xsd:complexType name="typeStudent">
    <xsd:attribute name="prenom"  type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="supervisor" type="typeResponsable"/>
  <xsd:complexType name="typeSupervisor">
   <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
 </xsd:schema>
```

a XML element

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<course intitule="M2PLS">
 <supervisor nom="Boudes"/>
 <student prenom="Jean" nom="Dupond"/><student prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Example : XML schema for courses (1) formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="course" type="typeCourse"/>
  <xsd:complexType name="typeCourse">
   <xsd:attribute name="title" type="xsd:string"  use="required"/>
   <xsd:sequence>
    <xsd:element ref="supervisor" />  <xsd:element ref="student"/>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="student" type="typeStudent"/>
  <xsd:complexType name="typeStudent">
    <xsd:attribute name="prenom"  type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="supervisor" type="typeResponsable"/>
  <xsd:complexType name="typeSupervisor">
   <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
 </xsd:schema>
```

a course:

has a mandatory attribute name

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<course intitule="M2PLS">
 <supervisor nom="Boudes"/>
 <student prenom="Jean" nom="Dupond"/><student prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Example : XML schema for courses (1) formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="course" type="typeCourse"/>
  <xsd:complexType name="typeCourse">
   <xsd:attribute name="title" type="xsd:string"  use="required"/>
   <xsd:sequence>
    <xsd:element ref="supervisor" />  <xsd:element ref="student"/>
   </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="student" type="typeStudent"/>
  <xsd:complexType name="typeStudent">
    <xsd:attribute name="prenom"  type="xsd:string" use="required"/>
    <xsd:attribute name="nom"
  </xsd:complexType>
  <xsd:element name="supervisor" type="typeResponsable" />
  <xsd:complexType name="typeSupervisor">
   <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
 </xsd:schema>
```

a course:

has a supervisor

includes students

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<course intitule="M2PLS">
 <supervisor nom="Boudes"/>
 <student prenom="Jean" nom="Dupond"/><student prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Example : Java code to/from XML data (2)

```
%xjc course.xsd -p "up13.course" -d .
// dans up13/formation/, javac *.java
%javac -cp .:$CLASSPATH *.java
```

```
import java.io.*;    import javax.xml.bind.*;    import up13.course.*;
public class Test {
  public static void main(String args[]) {
    try {
      JAXBContext jaxbContext = JAXBContext.newInstance ("up13.course");
      Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
// to verify wrt a XML schema
      unmarshaller.setEventHandler(new CourseValidationEventHandler());
      Course aCourse = (Course)
                    unmarshaller.unmarshal(new File("M2PLS.xml"));

      System.out.println("Course:␣" + aCourse.getTitle());
      Course.Supervisor supervisor = aCourse.getSupervisor();
      System.out.println("Supervisor:␣" + supervisor.getNom());

// generation of data
      Course anotherCourse = new Course();
      anotherCourse.setTitle("M2EID");
// XML generation
      Marshaller marshaller = jaxbContext.createMarshaller();
     marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
      marshaller.marshal( anotherCourse, System.out );
    } catch (JAXBException e) {System.err.println(e);}
} }
```

Example : Java code to/from XML data (2)

```
%xjc course.xsd -p "up13.course" -d .
//cans up13/formation/, javac *.java
%javac -cp .:$CLASSPATH *.java
```

creates

in this package (-p)

written in this folder (-d)

for this schema

a file ObjectFactory.java (intermediary between data and objects)
and a class for each type described in the schema (see next slide)

```
import java.io.*;          import up13.course.*;
public
  public static void main(String args[]) {

                           ontext = JAXBContext.newInstance ("up13.course");
                           rshaller = jaxbContext.createUnmarshaller();
                           schema
                           ventHandler(new CourseValidationEventHandler());
                           (Course)
                          unmarshaller.unmarshal(new File("M2PLS.xml"));

      System.out.println("Course: " + aCourse.getTitle());
      Course.Supervisor supervisor = aCourse.getSupervisor();
      System.out.println("Supervisor: " + supervisor.getNom());

// generation of data
      Course anotherCourse = new Course();
      anotherCourse.setTitle("M2EID");
// XML generation
      Marshaller marshaller = jaxbContext.createMarshaller();
     marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
      marshaller.marshal( anotherCourse, System.out );
    } catch (JAXBException e) {System.err.println(e);}
} }
```

Example : Java code to/from XML data (2)

```
%xjc course.xsd -p "up13.course" -d .
// dans up13/formation/, javac *.java
%javac -cp .:$CLASSPATH *.java
```

```
import java.io.                x.xml.bind.*;     import up13.course.*;
public class Test {
  public static void main(String args[]) {
    try {
      JAXBContext jaxbContext = JAXBContext.newInstance (up13.course.);
      Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
// to verify wrt a XML schema
      unmarshaller.setEventHandler(new CourseValidationEventHandler());
      Course aCourse = (Course)
                    unmarshaller.unmarshal(new File("M2PLS.xml"));

      System.out.println("Course: " + aCourse.getTitle());
      Course.Supervisor su                ervisor();
      System.out.println("                r.getNom());

// generation of data
      Course anotherCourse = new Course();
      anotherCourse.setTitle("M2EID");
// XML generation
      Marshaller marshaller = jaxbContext.createMarshaller();
     marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
      marshaller.marshal( anotherCourse, System.out );
    } catch (JAXBException e) {System.err.println(e);}
} }
```

context allowing the creation of :

- object analysing XML documents

analysis of a XML document and generation of a Java object in the VM

- object generating XML documents

Example : Java code to/from XML data (3)

```
package up13.course;

import javax.xml.bind.annotation.XmlAccessType;
...

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"supervisor","student"})
@XmlRootElement(name = "course")
public class Course {
    @XmlElement(required = true)
    protected Course.Supervisor supervisor;
    protected List<Course.Student> student;
    @XmlAttribute(name = "title")
    protected String title;

    /**
     * Gets the value of property supervisor.
     *
     * @return
     *     possible object is
     *     {@link Course.Supervisor }
     *
     */
    public Course.Supervisor getSupervisor() {
        return supervisor;
    }
...
```

Example : Java code to/from XML data (4)

```
import javax.xml.bind.*;

public class FormationValidationEventHandler implements ValidationEventHandler{

      public boolean handleEvent(ValidationEvent ve) {
        if (ve.getSeverity()==ValidationEvent.FATAL_ERROR ||
            ve .getSeverity()==ValidationEvent.ERROR){
            ValidationEventLocator  locator = ve.getLocator();
            //
            System.out.println("Invalid␣Course␣Document:␣:␣"
                               + locator.getURL());
            System.out.println("Error␣:␣" + ve.getMessage());
            System.out.println("column␣" +
                               locator.getColumnNumber() +
                               ",␣line␣"
                               + locator.getLineNumber());
         }
         return true;
       }
   }
```

Example : Java c ata (4)

Event manager for the XML analyser (errors, ...)

```
import javax.xml.bind.*;

public class FormationValidationEventHandler implements ValidationEventHandler{

      public boolean handleEvent(ValidationEvent ve) {
        if (ve.getSeverity()==ValidationEvent.FATAL_ERROR ||
            ve .getSeverity()==ValidationEvent.ERROR){
            ValidationEventLocator  locator = ve.getLocator();
            //
            System.out.println("Invalid␣Course␣Document:␣:␣"
                               + locator.getURL());
            System.out.println("Error␣:␣" + ve.getMessage());
            System.out.println("column␣" +
                               locator.getColumnNumber() +
                               ",␣line␣"
                               + locator.getLineNumber());
        }
        return true;
      }
   }
```

5 Tools : Messaging, name servers

- Mailing
- Name server

5 Tools : Messaging, name servers

- Mailing
- Name server

The mailing is an asynchronous I/O system.
Used in "ordinary" mailing, but also for asynchronous communicationbeteen processes.

- `javax.mail.*` is the library containing the necessary classes (optional package in Java SE, included in J2EE, see `https://javaee.github.io/javamail/`).
- Contains a generic API together with APIs for the various providers (or mail protocols).
- APIs for the following protocols are present in the package :
  - for sending mails : SMTP
  - for reading mails : IMAP4, POP3
- Language for some webmails

Output of mails :

- `Session`
  - defines the parameters for sending mails :
    - `mail.transport.protocol` : sendmail protocol (e.g. `"smtp"`)
    - `mail.smtp.host` : sendmail server
  - `getInstance(Properties prop)` creates an instance de session from properties (there may be a second argument for authentication : `Authenticator authenticator`)
- `MimeMessage` instances are object in MIME format together with session information
- `Transport` effective sender of a message

Example :

```
import java.util.*;
import javax.mail.*; import javax.mail.internet.*;

public class SendMail {
   public static void main (String[] args) throws MessagingException
   {
      Properties props = System.getProperties();

      props.put("mail.transport.protocol", "smtp");
      props.put("mail.smtp.host", "upn.univ-paris13.fr");
      // or -Dmail.transport.protocol=smtp ...

      Session session = Session.getInstance(props);

      MimeMessage message = new MimeMessage(session);
      message.setFrom(new InternetAddress("moi@univ-paris13.fr"));
      message.setRecipient(
         Message.RecipientType.TO,         // or CC ou BCC
         new InternetAddress("une.personne@univ-paris13.fr")
      );
      message.setSubject("Test");
      message.setText("ceci␣est␣le␣contenu␣du␣test␣!");

      Transport.send(message);
   }
}
```

To retrieve the mail server from his/her environment (linux/unix) :
if his/her mail address is, e.g. `univ-paris13.fr`,

```
% nslookup
> set query=MX
> univ-paris13.fr
...
univ-paris13.fr mail exchanger = 100 upn.univ-paris13.fr.
```

Reading of mails :

- `Session`
  - same as before with the following property :
    - `mail.store.protocol` : reading protocol (e.g. `"imap"`)
  - `getInstance()` same as before
- `Store` mail account
- `Folder` mail folder for an account
- `Message` message in a mail folder

Example :

```
import java.util.*;
import javax.mail.*;

public class ReadMail {

   public static void main (String[] args) throws Exception
   {
      Properties props = System.getProperties();

      Session session = Session.getInstance(props, null);

      Store store = session.getStore("imap");
      store.connect("imap.univ-paris13.fr","christophe.fouquere","monPassword");

      Folder inbox = store.getFolder("INBOX");
      inbox.open(Folder.READ_ONLY);

      Message message = inbox.getMessage(1); // le 1er message
      message.writeTo(System.out);

      inbox.close(false);
      store.close();
   }
}
```

5 Tools : Messaging, name servers

- Mailing
- Name server

Naming server allow for retrieving properties from keys :

- File systems : ext3, NTFS, ...
- Domain name system : DNS
- Directories : NIS, LDAP, Active Directory, ...

A **context** is a set of key-value pairs (*bindings*) :

- keys are organized in a tree structure :
  - `/usr/bin` is a sub-context of `/usr` (ext3)
  - `univ-paris13.fr` is a sub-context of `fr` (DNS)
  - `ou=structures,dc=univ-rennes1,dc=fr` is a sub-context of `dc=univ-rennes1,dc=fr` (LDAP)
- the following operations may be done on contexts :
  - bind a value to a key
  - delete a binding
  - list the bindings

The necessary software is defined by :

- an API `javax.naming.*` for programming, methods call a SPI
- a SPI that depends on the type of the server (*Service Provider Interface*)
- the interface `Context` that defines the abstract structure

The initial object is given by :

- `InitialContext` initialized with :
  - a SPI given as value for the attribute `naming.factory.initial`
  - a root

The SPI is service dependent, it is a class loaded during the execution.

- `com.sun.jndi.fscontext.RefFSContextFactory` is the SPI given by Java for file systems.

Standard Operations :

- `NamingEnumeration<NameClassPair> list(String name)`
- `void rename(String oldName, String newName)`
- `Context createSubContext(String name)`
- `Object lookup(String name)`

The interface `NamingEnumeration<T>` declares the following methods : `close()`, `hasMore()`, `next()`
The class `NameClassPair` defines a binding.

```
import java.util.*;
import javax.naming.*;

public class SGF_jndi {
   public static void main(String args[]) throws Exception {
      Properties props = new Properties();
      props.put("java.naming.factory.initial",
                        "com.sun.jndi.fscontext.RefFSContextFactory");
      props.put("java.naming.provider.url", "file:/");

      Context ctx = new javax.naming.InitialContext (props);

      NamingEnumeration ne = ctx.list("etc");
      while (ne.hasMore()) System.err.println(ne.next());
   }
}
```

6 Client-server : control

Client-server

What kinds of massive attacks ?

- spam to a mail server
- denial of service (server stack)
- robot on forms

How can we hedge against attacks ? 2 means

- Distinguish human vs program
- Distinguish “true” client vs “false” client

There is no 100-percent effective policies.

Human versus program

Captcha :
(completely automated public Turing test to tell computers and humans apart, 2000, CMU)
Distortion of letters and digits for a hard automatic recognition :

- distortion of each letter
- addition of colors
- adhesion of letters
- highlighting
- audio version

In Java, several APIs :

- JCaptcha (see next)
- SimpleCaptcha
- reCAPTCHA

Part of a jsp :

```
...
<h2>Using JCaptcha: A Simple Captcha Servlet</h2>
<form name="SimpleServletForm" action="/captcha-demos/simple" method="post">
<input type="hidden" name="hidCaptchaID" value="<%=␣session.getId()␣%>"/>
<table border="0" cellspacing="0" cellpadding="3">
  <tr>
    <td valign="middle">Enter these letters:<br/>
    <img src="/captcha-demos/simpleCaptchaServlet" align="middle" border="1"/></td>
    <td><input type="text" name="inCaptchaChars"/></td>
  </tr>
  <tr>
    <td colspan="2" align="right"><input type="submit" value="Submit"/></td>
  </tr>
</table>
</form>
...
```

with a mapping `captcha-demos/simpleCaptchaServlet` onto the class of `SimpleCaptchaServlet.java`

```
import com.octo.captcha.service.CaptchaServiceException;
import javax.servlet.*;
import javax.servlet.http.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Map;
import javax.imageio.ImageIO;

public class SimpleCaptchaServlet extends HttpServlet {
  String sImgType = null;
  public void init( ServletConfig servletConfig ) throws ServletException
  {
    super.init( servletConfig );

    // For this servlet, supported image types are PNG and JPG.
    sImgType = servletConfig.getInitParameter( "ImageType" );
    sImgType = sImgType==null ? "png" : sImgType.trim().toLowerCase();
    if ( !sImgType.equalsIgnoreCase("png") && !sImgType.equalsIgnoreCase("jpg") &&
    !sImgType.equalsIgnoreCase("jpeg") )
    {
      sImgType = "png";
    }
  }
....
```

```
....
  protected void doGet( HttpServletRequest request, HttpServletResponse response )
  throws ServletException, IOException
  { ByteArrayOutputStream imgOutputStream = new ByteArrayOutputStream();
    byte[] captchaBytes;

    try {
      // Session ID is used to identify the particular captcha.
      String captchaId = request.getSession().getId();

      // Generate the captcha image.
      BufferedImage challengeImage = MyCaptchaService.getInstance()
               .getImageChallengeForID(captchaId, request.getLocale() );
      ImageIO.write( challengeImage, sImgType, imgOutputStream );
      captchaBytes = imgOutputStream.toByteArray();

      // Clear any existing flag.
      request.getSession().removeAttribute( "PassedCaptcha" );
    } catch( Exception e ) {  ... }

    // Set appropriate http headers.
    response.setHeader( "Cache-Control", "no-store" );
    response.setHeader( "Pragma", "no-cache" );
    response.setDateHeader( "Expires", 0 );
    response.setContentType( "image/"+(sImgType.equalsIgnoreCase("png")?"png":"jpeg"));

    // Write the image to the client.
    ServletOutputStream outStream = response.getOutputStream();
    outStream.write( captchaBytes );
    outStream.flush();
    outStream.close();
  }
....
```

```
....
  protected void doPost( HttpServletRequest request, HttpServletResponse response )
  throws ServletException, IOException {
    // Get the request params.
    Map paramMap = request.getParameterMap();
    String[] arr1 = (String[])paramMap.get( "hidCaptchaID" );
    String[] arr2 = (String[])paramMap.get( "inCaptchaChars" );
    String sessId = request.getSession().getId();
    String incomingCaptchaId = arr1.length>0 ? arr1[0] : "";
    String inputChars = arr2.length>0 ? arr2[0] : "";

    // Check validity and consistency of the data.
    if ( sessId==null || incomingCaptchaId==null || !sessId.equals(incomingCaptchaId) )
    { response.sendError( HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "No cookies." );
      return; }

    // Validate whether input from user is correct.
    boolean passedCaptchaTest = validateCaptcha( incomingCaptchaId, inputChars );

    // Set flag into session.
    request.getSession().setAttribute( "PassedCaptcha", new Boolean(passedCaptchaTest) );

    // Forward request to results page.
    RequestDispatcher rd = getServletContext().getRequestDispatcher( "/results.jsp" );
    rd.forward( request, response );
  }
....
```

```
....
  private boolean validateCaptcha( String captchaId, String inputChars ) {
    boolean bValidated = false;
    try {
      bValidated = MyCaptchaService.getInstance()
                .validateResponseForID( captchaId, inputChars );
    } catch( CaptchaServiceException cse ) { ... }
    return bValidated;
} }
```

```
import com.octo.captcha.service.image.ImageCaptchaService;
import com.octo.captcha.service.image.DefaultManageableImageCaptchaService;

public class MyCaptchaService
{
  // a singleton class
  private static ImageCaptchaService instance =
                    new DefaultManageableImageCaptchaService();
  public static ImageCaptchaService getInstance()
  {
    return instance;
  }
}
```

Program versus program

Idea : test if the client can compute quickly ! (1998)

Protocol : **Proof-of-work**

Client side :
- lookup of a string X such that the the hash of the string “<date> :<address> :X” begins with 20 (ou 50, ...) zero-bits
- send of X and the hash to the server

Server side :
- test X and the hash sent by the client

Mechanism used in the following cases :

- mailer antispam (*Hashcash*)
- secure transactions in distributed systems (*bitcoin*)

bitcoin : transaction system for virtual currency

- an address = a public key
- the private key is the cryptographic element guaranteeing that who has it is the owner of the bitcoins
- it consists in a peer-to-peer network
- each host loads the set of transactions already done
- transaction requests are gathered
- a host is randomly chosen to prove that transactions are correct (10mn)

7 Security policy in Java

- Introduction
- java.security
- java.policy

7 Security policy in Java

- Introduction
- java.security
- java.policy

Security Policy in Java

(Security Policy)

A **security policy** determines permissions and rights to Java code external to the VM and to connections from clients.

Two main classes :

- java.lang.Policy : to specify the security policy
- java.lang.SecurityManager : to apply the security policy

By default : no security policy !
If a security manager is loaded :

- For each potentially dangerous method, a call is sent to the security manager to test it wrt the security policy

Examples :
– checkRead() : tests if a thread has the right to read a particular file,
– checkWrite() : tests if a thread has the right to write in a file.

Lists of potentially dangerous operations :

- modification or access to system properties
- modification to a code, a thread or a process (priority, exit, ...)
- request for a socket connection, launching of a server
- file management
- creation of a class loader
- dynamic load of classes
- modification of packages

What is not tested :

- creation of threads (DoS)
- memory management (DoS)
- some applet operations : sending of messages, data control

2 methods for defining security policies :

- Inside your Java code
- Outside your java code by means of policy files

Inside the code

```
import java.io.*;

public class TestSecurity {
  public void test (File f) {
   try {
     FileInputStream fis = new FileInputStream(f);
     int b;
     while ((b = fis.read()) != -1);
     fis.close();
   }
   catch (Exception e) {System.out.println(e);}
  }

  public static void main(String args[]) {
   FilePermission perm = new FilePermission("TestSecurity.java",
                                            "write");
   SecurityManager sm = System.getSecurityManager();
   if (sm != null) sm.checkPermission(perm);
   else {System.out.println("sm␣??");};

   (new TestSecurity()).test(new File("TestSecurity.java"));
  }
}
```

Test with :

```
java -Djava.security.manager TestSecurity
```

Outside the code

General folder for files describing the security policy :

```
<chemin>/jdk/jre/lib/security
```

it contains :

- `java.security` : main file ! !
- `java.policy` : main file for the security policy
- `javaws.policy` : Java Web Start policy
- `trusted.libraries` : trusted libraries
- `blacklist` : blacklisted signatures
- `cacerts` : accepted certificates
- `policy/limited/local_policy.jar` : policy for cipher sizes
- `/policy/limited/US_export_policy.jar` : policy for cipher sizes in USA

7 Security policy in Java

- Introduction
- java.security
- java.policy

java.security

File containing a list of pairs key-value and comments

```
#
# Class to instantiate as the system Policy.
# This is the name of the class
# that will be used as the Policy object.
#
policy.provider=sun.security.provider.PolicyFile

# The default is to have a single system-wide policy file,
# and a policy file in the user's home directory.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

And other informations :

- classes for cipher providers
- restriction on certificates
- list of packages with restricted control

7 Security policy in Java

- Introduction
- java.security
- java.policy

java.policy

```
// Standard extensions get all permissions by default
grant codeBase "file:${{java.ext.dirs}}/*" {
    permission java.security.AllPermission;
};

// default permissions granted to all domains
grant {
    // Allows any thread to stop itself with java.lang.Thread.stop()
    permission java.lang.RuntimePermission "stopThread";

    // allows anyone to listen on un-privileged ports
    permission java.net.SocketPermission "localhost:1024-", "listen";
};
```

```
// default permissions granted to all domains
grant {
    // "standard" properies that can be read by anyone
    permission java.util.PropertyPermission "file.separator", "read";
    permission java.util.PropertyPermission "path.separator", "read";
    permission java.util.PropertyPermission "line.separator", "read";

    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.vendor", "read";
    permission java.util.PropertyPermission "java.vendor.url", "read";
    permission java.util.PropertyPermission
            "java.class.version", "read";
    permission java.util.PropertyPermission "os.name", "read";
    permission java.util.PropertyPermission "os.version", "read";
    permission java.util.PropertyPermission "os.arch", "read";
};
```

```
// default permissions granted to all domains
grant {
    // "standard" properies that can be read by anyone
    permission java.util.PropertyPermission
            "java.specification.version", "read";
    permission java.util.PropertyPermission
            "java.specification.vendor", "read";
    permission java.util.PropertyPermission
            "java.specification.name", "read";

    permission java.util.PropertyPermission
            "java.vm.specification.version", "read";
    permission java.util.PropertyPermission
            "java.vm.specification.vendor", "read";
    permission java.util.PropertyPermission
            "java.vm.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.version", "read";
    permission java.util.PropertyPermission "java.vm.vendor", "read";
    permission java.util.PropertyPermission "java.vm.name", "read";
};
```

java.policy : format for each entry

```
grant signedBy "signer_names", codeBase "URL",
        principal principal_class_name "principal_name",
        principal principal_class_name "principal_name",
        ... {

      permission permission_class_name "target_name", "action",
          signedBy "signer_names";
      permission permission_class_name "target_name", "action",
          signedBy "signer_names";
      ...
  };
```

java.policy : format for each entry

- `signedBy` : list of certificate aliases in the keystore
- `codeBase` : specifies from which host/folder a code may come from (by default : from anywhere)
- `principal` : specifies a user name
- `permission` : `permission_class_name` should be a subclass of `Permission`, list authorized permissions.

Standard subclasses of `java.security.Permission`

- `AllPermission` : accepts anything (dangerous !)
- `RuntimePermission` : for system control (loader, thread, ...)
- `FilePermission` : read, write, execute, delete, readlink
- `MBeanPermission` : management of beans (JEE)
- `PrivateCredentialPermission` : access control on names (cf. Principal)
- `ServicePermission` : access control on authorizations
- `SocketPermission` : access control on sockets (accept, connect, listen, resolve)
- `UnresolvedPermission`

java.policy : format of an entry

for the codeBase : specifies where is loadable classes

- necessarily an URL
- forme `...truc` : file `truc`
- forme `...truc/` : in folder `truc` except files `trux/xx.jar`
- forme `...truc/*` : in folder `truc` with files `trux/xx.jar`
- forme `...truc/-` : recursively in folder `truc`.

Launching of the security manager :

```
java -Djava.security.manager monCode
```

To add a security policy file :

```
java -Djava.security.manager -Djava.security.policy=uneURL monCode
```

To replace the standard security policy file :

```
java -Djava.security.manager -Djava.security.policy==uneURL monCode
```