

Programmation distribuée Java

Christophe Fouqueré

Master Informatique 1ère année

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

Bibliographie (très) partielle :

- **Java La synthèse, Des concepts objet aux architectures Web**, Clavel et alii, Dunod Informatiques, 2000
- **Programmation réseau avec Java**, E. Rusty Harold, O'Reilly, 2001
- et bien sûr `http://docs.oracle.com`
en particulier
`http://docs.oracle.com/javase/6/docs/api/`
ou `http://docs.oracle.com/javase/7/docs/api/`

1

Introduction

2

Threads : architectures maître-esclave

- Processus légers
- Exemples types
- Pools de processus légers

3

Réseau : adressage et sockets

- Connexions réseau
 - Rappels sur les réseaux (IP, TCP et UDP, adressage)
 - Manipulation d'adresses : classe `InetAddress`
 - Connexions TCP non sécurisés entre machines (classe `Socket`)
 - Connexions TCP sécurisés entre machines (classe `SSLServerSocket`)
 - Connexions UDP entre machines (classe `DatagramSocket`)
 - Connexions entre machines (classe `MulticastSocket`)
- URL
 - Classes `URL` et `URLConnection`
 - Gestion de protocole
- RMI et RMI-IIOP
 - RMI : Remote Method Invocation

4

Gestion d'annotations

5 Outils : messagerie, serveur de noms

- Messagerie
- Serveur de noms

6 Client-serveur : contrôle

7 Politique de sécurité en Java

- Introduction
- java.security
- java.policy

1 Introduction

Objectifs du cours :

- Usage d'environnements de développements (Eclipse, Netbeans) : apprentissage en TP
- Compréhension des pratiques logicielles en environnement ouvert : sockets, url, serveur de nommage
- Applications : mail, rmi, ...

- 1969 : Arpanet
- 1979 : Internet
- 1974-1982 : TCP
- 1991 : Web
- 1989-1996 : standard CORBA (Common Object Request Broker Architecture) de l'OMG (Object Management Group)
- 1990-1996 : modèle COM (Component Object Model) puis DCOM puis ActiveX de Microsoft
- 1994 : Java
- 1997 : 1er "web service", plateforme Jini de Sun
- 1998- : Développement d'un ensemble de protocoles par standardisation (W3C, OMG, groupes de constructeurs)
- 2000- : Protocoles Web2.0, BPEL, ...

Java : Une classe = structure + méthodes


```

package test;
import java.util.*;

public class Personne {
    private String nom;
    protected List<String> prenom;
    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 String prenom;
    public String societe;
    private 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";
    }
}

```

Annotations:

- `package test;`: nom du paquetage (donc test.Personne)
- `import java.util.*;`: importation de classes
- `public class Personne {`: définition de classe (convention: début majuscule)
- `Personne(Object o) {`: constructeur de classe (donc pas de constructeur vide)

```

$ 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;
    utilisable localement 1.*;

public class Personne {
    utilisable par héritage
    private String nom;
    protected List<String> prenom;
    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) {
    utilisable globalement 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 non_sérialisable {
    private String nom;
    protected List<String> prenom;
    public transient String societe;
    private static final String societeException
        = new String("inconnu");

    public boolean defini;
    public variable_de_classe ifiant() {System.out.println
        ("je_m'appelle" + nom);}

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

    public static test_d'instance in(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> prenom;
    public transient String societe;
    private static final String societeException = "societe inconnu";

    public boolean defini;
    public void identifiant() {
        // ...
        ("je_m'appelle" + nom);
    }

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

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

```

type paramétré

type primitif (comme byte, char, double, float, int, long, short)

méthode principale pour un programme

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 : Rappels : interfaces

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

```
import java.io.*;  
  
public class DigestServer implements DigestListener {  
    ...  
    public void setDigest(byte[] digest) {  
        ...  
    }  
    ...  
}
```

Java : Rappels : interfaces

interface = signature de (partie de) classe

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

```
import java.io.*;  
  
public class DigestServer implements DigestListener {  
    ...  
    public void setDigest(byte[] digest) {  
        ...  
    }  
    ...  
}
```

classe implantant les méthodes de l'interface

Java : Rappels : classes abstraites et 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 : Rappels : classes abstraites et 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();  
    }  
}
```

classe abstraite,
méthode abstraite

extension de classe

```
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 : Rappels : classes abstraites et extensions

```
abstract class Triangle { surcharge de méthode
    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 : Rappels : gestion des 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 : Rappels : gestion des exceptions

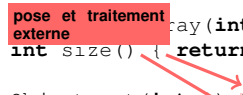
```
class ExpandableArray {
    protected Object[] data;
    protected int size = 0;

    public pose et traitement externe array(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 {};
```



```

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();

            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);
        }
        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-paris13.fr",80);
            Writer output = new OutputStreamWriter
                (connexion.getOutputStream(), "8859_1");

            output.write("GET_/_HTTP_1.0\r\n\r\n"); output.flush();

            test et traitement local

            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);
        }
        catch (IOException e) {System.out.println(e);}
        finally {
            try {if (connexion != null) connexion.close();}
            catch (IOException e) {System.out.println(e);}
        } } }
```

- Assertions :

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

devient

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

Types génériques :

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

devient

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


Itérations :

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

Itérations :

La variable *i* parcourt *a*

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

- Boxing (i.e. passage automatique entre types primitifs et classes associées, e.g. `int` et `Integer`).
- Type énumératif :

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

- Nombre d'arguments variable :

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

Annotations : permet aux logiciels intermédiaires (compilateurs, interpréteurs, environnements, ...) d'effectuer des tests, des vérifications, des ajouts de code.

Déclaration d'une annotation :

```
import java.lang.annotation.*;  
  
@Retention(RetentionPolicy.RUNTIME)  
@Target(ElementType.METHOD)  
public @interface Test { }
```

Déclaration du code source :

```
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() { }  
}
```

Déclaration du code intermédiaire :

```
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);
    }
}
```

Différentes bibliothèques supplémentaires ou améliorées :

- JAX-WS : web services
- JDBC
- Java compiler API
- Annotations prédéfinies supplémentaires

Scripting / Java :

```
import javax.script.*;
public class ScriptTest{
    public static void main(String[] args){
        try{
            // Create an instance of the Scripting manager.
            ScriptEngineManager manager = new ScriptEngineManager();

            // Get the reference to the rhino scripting engine.
            ScriptEngine engine = manager.getEngineByName
                ("javascript");

            // Get the Binding object for this Engine.
            Bindings bindings = engine.getBindings
                (ScriptContext.ENGINE_SCOPE);

            // Put the input value to the Binding.
            bindings.put("strValue", "A_Test_String");
```



```
// Populate the script code to be executed.
StringBuilder scriptCode = new StringBuilder();
scriptCode.append("var_javaString
======_new_java.lang.String(strValue);");
scriptCode.append("var_result
======_javaString.length();");

// Evaluate the Script code.
engine.eval(scriptCode.toString());

// Take the output value from the script, i.e Bindings.
int strLength = (Integer)bindings.get("result");

System.out.println("Length_is_" + strLength);
} catch (Exception exception) {
    exception.printStackTrace();
}
}
}
```

Modifications diverses :

```
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<>();
```

et aussi pointeurs vers des fonctions, ... (cf. `java.lang.invoke`)

- Expressions lambda
- annotations améliorées
- `java.util.stream`
- sécurité et diverses autres choses

Syntaxe d'une 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());
```

Possibilité d'utiliser des variables externes au corps et des arguments de la lambda seulement si ces variables sont inchangées dans la suite du programme (comme les variables **static** par exemple).

Le type d'une lambda-expression est une **interface fonctionnelle** : interface à méthode unique (ou presque ...) :

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

Utilisation dans un 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);
    }
}
```

Possibilité d'accéder directement aux méthodes (vues alors comme des lambda-expressions nommées) :

```
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 );
    }
}
```

```
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;
    }
}
```

Un `Stream` permet effectivement d'appliquer des fonctions sur un flux d'éléments.

Un flux peut être créé à partir de collections, de listes, ... avec la méthode `stream()`.

Un flux permet de modifier des objets à la volée.

Les fonctions de modification permettent d'itérer, d'appliquer, de concaténer des résultats, ...

Les arguments fonctionnels de ces fonctions de modification ont l'annotation `FunctionalInterface` et l'un des types (ou des variantes) :

- `Function<T,R>` : prend un objet de type `T` et retourne un objet de type `R`
- `Supplier<T>` : retourne un objet de type `R`
- `Predicate<T>` : retourne un booléen selon l'objet de type `T`
- `Consumer<T>` : effectue une opération sur l'objet de type `T`

Exemple avec 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());
    }
}
```

avec la classe Personne (slide suivant)

```
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);  
    }  
}
```

Autre exemple avec Predicate et 1 Stream parallèle :

```
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());
    }
}
```

Autre exemple avec Predicate et 1 Stream séquentiel :

```
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);
    }
}
```

2 Threads : architectures maître-esclave

- Processus légers
- Exemples types
- Pools de processus légers

2 Threads : architectures maître-esclave

- Processus légers
- Exemples types
- Pools de processus légers

Package : `java.lang.Thread`

distribution (locale) des calculs à effectuer :

1 tâche effectuée par une unité de calcul avec

autonomie mémoire totale	==>	processus
partage de la mémoire	==>	thread

Exécution d'un processus léger :

création	==>	<code>thread()</code>
phase d'initialisation	==>	<code>start()</code>
phase de calcul	==>	<code>run()</code>
phase d'arrêt	==>	<code>interrupt()</code> (ou <code>stop()</code> redéfini)
destruction	==>	géré par le garbage collector

Partage de mémoire entre threads :

- **en phase d'initialisation :**

- certaines variables doivent être en pot commun alors que d'autres sont réservées au calcul du thread

==> **volatile**

- **en phase de calcul :**

- des calculs peuvent nécessiter la terminaison d'autres calculs, ou l'attente de données ou d'une date

==> `interrupt()` / `join()` / `sleep()`

- des calculs peuvent porter sur la même zone de données (problème de lecture/écriture partagée)

==> **synchronized**

Classes Thread et 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();  
...
```

- instance de Thread = contrôleur de processus léger.
- à l'initialisation : un unique thread lançant `main()` de la classe appelée.
- classe Thread
 - **static** Thread `currentThread()`
retourne le thread courant (i.e. l'objet instance)
 - **void** `setName(String s)`
permet de donner un nom au Thread (utilisé par `toString()`)
(par défaut `Thread-i` et `Thread-0` pour le main)
- c'est l'appel à `start()` qui crée vraiment un processus léger.
- un appel à `run()` direct exécute `run()` sans créer de nouveau processus.

```

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");
    }
}

```

- **Priorité :**

- **entre** `Thread.MIN_PRIORITY=1` et `Thread.MAX_PRIORITY=10`
- **par défaut** `Thread.NORM_PRIORITY=5`
- **classe** `Thread`
 - **int** `getPriority()` renvoie la priorité du `Thread`
 - **void** `setPriority(int priority)` positionne la priorité

- **Protection via un objet** (au plus un thread exécutant). A chaque objet est associé un descripteur contenant des informations sur cet objet. La synchronisation passe par le mécanisme suivant :
 - une information particulière (un drapeau) indique si l'objet est en cours d'utilisation synchronisée.
 - l'exécution d'une instruction (ou d'un groupe, ou d'une méthode) peut être conditionnée par ce drapeau

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

- on peut prendre `this` comme objet de synchronisation (c'est implicitement ce qui est fait lors d'une synchronisation sur méthode)
- les méthodes sont synchronisables sauf :
 - les méthodes d'interface
 - les constructeurs
- la spécification de synchronisation n'est pas héritable
- la synchronisation de champs statiques est possible sur les objets-classes correspondants
(... **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 {}

```

Gestion mémoire locale / partagée

- chaque instance (de `Runnable`) a ses propres variables
- tous les threads lancés sur la même instance partagent ces variables
- le modifieur `volatile` force la synchronisation des variables partagées entre threads

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();
    } }
```

A l'exécution :

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

Données locales à un thread : classe `ThreadLocal`

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

Exemple :

```
public class R implements Runnable {  
    Object o; // variable partagée  
  
    ThreadLocal v = new ThreadLocal();  
                // variable locale  
  
    ...  
    o = ...;  
    v.set(u);    // où u est un objet quelconque  
    ...  
    .. = .. o ..;  
    .. = .. v.get() ..;  
}
```

Groupe de threads

- classe `ThreadGroup` (par défaut à la création du thread, le groupe est celui du créateur) :
 - `ThreadGroup ThreadGroup(String s)` crée un threadgroup de nom `s`
 - `void setMaxPriority(int priority)` fixe la valeur maximale de priorité du groupe
- classe `Thread` :
 - `ThreadGroup getThreadGroup()` renvoie le threadgroup du thread
 - `Thread Thread(ThreadGroup g, ImplRun r, String s)` crée un nouveau thread dans le groupe `g` avec le Runnable `r` (`ImplRun` doit implanter `Runnable`) et le nom `s`

2 Threads : architectures maître-esclave

- Processus légers
- **Exemples types**
- Pools de processus légers

1. Callback après appel de thread (méthode par continuation)

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

public class CbDigest implements Runnable {
    private File input;                // fichier à signer
    private CbDigestServer cbDigestServer; // objet demandeur

    public CbDigest (File input, CbDigestServer cb) {
        this.input = input;
        this.cbDigestServer = cb;        // déclaration
    }
    ...
}
```

```

...
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();
        cbDigestServer.setDigest(digest); // continuation
    }
    catch(IOException e) {System.err.println(e);}
    catch(NoSuchAlgorithmException e) {System.err.println(e);}
}
}

```

```

// Serveur
import java.io.*;

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

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

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

    void setDigest(byte[] digest) {             // méthode de reception
        this.digest = digest;
        System.out.println(this);
    }
    ... // cf page suivante

```

```

...
    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]);
            CbDigestServer d = new CbDigestServer(f);
            d.calculateDigest();
        }
    }
}

```


2. Callback après appel de thread (écouteurs)

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

public class CbDigest implements Runnable {
    private File input;
    private Vector digestList = new Vector(); // liste d'abonnés

    public CbDigest (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); // diffusion
        ...
    }
}
```

```
// Interface Listener
```

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

```
// Serveur
```

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

```
public class CbDigestServer implements DigestListener {  
    ...  
    public void setDigest(byte[] digest) { //méthode de réception  
        ...  
    }  
    ...  
}
```

3. Attente entre threads : `join()`

```
// Esclave
public class CbDigest extends Thread {
    ...
    public byte[] getDigest() {
        ...
    }
    ...
}
```

```
// Serveur

import java.io.*;

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

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

        for (i=0;i<args.length;i++) {
            try {
                cbDigestTab[i].join();        // blocage jusqu'à ce
                                                // que le thread termine
                byte[] digest = cbDigestTab[i].getDigest();
                String fileName = cbDigestTab[i].getFileName();
                System.out.print(fileName + ":_");
                ...
            } } } }

```

4. Attente entre threads via ressources : `wait()`

Remarques :

- `wait()` et `notify()` sont des méthodes de `Object`
- `wait()` entraîne une attente de levée du verrou
- `wait()` à l'intérieur d'un `synchronized`
- Versions de `wait()` :
 - `wait()`
 - `wait(long millisec)`
 - `wait(long millisec, int nanosec)`
- Versions de `notify()` :
 - `notify()` réveil d'un thread
 - `notifyAll()` réveil de tous les threads

```
// Esclave

import java.io.*; import java.security.*;

public class CbDigest implements Runnable {
    private File input;
    private byte[] digest;
    private CbDigestServer server; // objet de notification

    public CbDigest (File input, CbDigestServer server) {
        this.input = input;
        this.server = server;
    }

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

```
// Serveur
```

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

```
public class CbDigestServer {
```

```
    private File input;
```

```
    private byte[] digest;
```

```
    public CbDigestServer(File input) {
```

```
        this.input = input;
```

```
    }
```

```
    public void setDigest(byte[] digest) {
```

```
        this.digest = digest;
```

```
    }
```

```
    ...
```

```
    public void calculateDigest() {
```

```
        synchronized(this) {
```

```
            CbDigest cb = new CbDigest(input, this); // this=objet d'attente
```

```
            Thread t = new Thread(cb);                // appel du thread
```

```
            t.start();
```

```
            try {
```

```
                wait();
```

```
            }
```

```
            catch (InterruptedException e) {
```

```
            }
```

```
            System.out.println(this); // la signature est maintenant connue
```

```
    }}
```

2 Threads : architectures maître-esclave

- Processus légers
- Exemples types
- Pools de processus légers

`java.util.concurrent.ExecutorService :`

- **static** `ExecutorService newCachedThreadPool()` Création d'un pool de threads à la demande avec réutilisation des threads libres.
- **static** `ExecutorService newFixedThreadPool(int nThreads)` Création d'un pool de taille max fixée de threads à la demande avec réutilisation des threads libres.
- **static** `ExecutorService newSingleThreadExecutor()` Création d'un pool de taille 1 de threads ==> Pile de longueur quelconque de demandes d'exécution.
- **void** `shutdown()` Fini les tâches en cours et stoppe le pool.
- **void** `execute(Runnable command)` Demande l'exécution de la commande par un thread du pool.
- `<T> Future<T> submit(Callable<T> task)` Demande l'exécution de la tâche par un thread et retourne une instance de `Future`

`java.util.concurrent.Callable<V>` : similaire à un `Runnable` mais permet de retourner une valeur,

- `V call()`

`java.util.concurrent.Future<V>` : pointeur vers un objet de calcul asynchrone,

- **boolean** `cancel(boolean mayInterruptIfRunning)`
Attempts to cancel execution of this task.
- `V get(long timeout, TimeUnit unit)` Attend timeout unités de temps et renvoie le résultat (si temps dépassé ou autre alors exception).

```

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 Création du pool ze)
        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); }
    }
}
```

Lancement d'un thread
via le 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);}   
    }  
}
```

Obligatoire car utilisé
comme Thread

3 Réseau : adressage et sockets

- Connexions réseau
- URL
- RMI et RMI-IIOP

3 Réseau : adressage et sockets

- Connexions réseau
- URL
- RMI et RMI-IIOP

Rappels sur les réseaux

- **IP** : *Internet Protocol*

- seul protocole de la couche réseau reconnu par Java (donc pas IPX et Appletalk)
- données transitant par datagrammes (en-tête + données)
- adressage IPv4 par 4 octets (IPv6 sur 16)

- **DNS** : *Domain Name System*

- traduction nom symbolique - adresse IP
- Ex. (cf. `nslookup`) : `www.univ-paris13.fr` 192.33.182.1

- **TCP** : *Transmission Control Protocol*

- reconstitution des paquets + ack

- **UDP** : *User Datagram Protocol*

- ni vérification de l'ordre, ni réexpédition

- **port** : (0 < port < 65535)

- association logique à un service 'adresse IP + port'
- Exemples (cf `/etc/services`) :
 - 21 ftp
 - 25 smtp
 - 80 HTTP (web par défaut)
 - 1099 RMI registry

- **URI** : *Uniform resource Identifier*

- adresse d'une ressource
- *schéma : syntaxe propre au schéma*
- en général : `schéma://autorité/chemin?requête` où
 - *autorité* = adresse destinataire
 - *chemin* = chemin "dans" cette adresse (en fait, réinterprété par l'autorité)
 - *requête* = données ou requête

- **URN** : *Uniform Resource Name*

- Exemple : `urn:isbn:1234567890`
- gestion de données par domaine de noms,
- récupération de la donnée via un serveur

- **URL** : *Uniform Resource Locator*

- de la forme

`protocole://login:passwd@autorité:port/chemin#section?requête`

- **file:** //<host>/<path> fichier sur disque local
 - file:///etc/services
 - <host> par défaut machine locale
- **ftp:** //<user>:<pwd>@<machine>:<port>/chemin;type=<typecode>
 - ftp://ftp.univ-mrs.fr/f.txt fichier distant (transfert)
 - <user> par défaut anonymous
 - <pwd> par défaut adresse électronique
 - <port> par défaut 21
 - <typecode> :
 - d lister un répertoire
 - a transfert ASCII
 - i transfert binaire
- **http:** //<machine>:<port>/<path>?<requete>
 - http://www.univ-paris13.fr (par défaut complété à index.html)
- **mailto:** <adreesee-mail> envoi de courriers
 - mailto:christophe.fouquere@univ-paris13.fr
- **imap:** //<user>:<passwd>@<host>/<chemin> serveur de messagerie
 - imap://login@adresseMachine/repertoire
 - pop3://login@adresseMachine/repertoire
- **telnet:** //<user>:<passwd>@<machine>:<port>/
 - telnet://F205-3/ connexion telnet

(exception générée : `UnknownHostException`)

Adresses IP et URL

une instance de `InetAddress` contient les informations :

- adresse symbolique d'une machine
- adresse IP

"constructeurs" (en fait appel du DNS local) :

- **static** `InetAddress getByName(String host)` 1^{re}info
- **static** `InetAddress[] getAllByName(String host)` toutes
- **static** `InetAddress getLocalHost()`

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

Remarques :

- on ne peut pas "créer" une structure type adresse IP
- `equals()` réécrit de telle manière que l'égalité soit testée sur l'adresse IP

Exemple :

```
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`)

Connexions TCP non sécurisés

Côté client : la classe `Socket` gère les connexions

- `Socket(String host, int port)`
 - ouvre une connexion avec `host + port`
 - si `host` n'a pas de serveur en écoute sur le port
 - alors retour avec une `IOException`
- `Socket(InetAddress host, int port)`
- `Socket(String host, int port, InetAddress interface, int portLocal)`
 - permet de spécifier la partie source (par défaut le port source est le 1^{er} libre)

- `InetAddress getAddress()`
- **int** `getPort()`
- **int** `getLocalPort()`
- `InetAddress getLocalAddress()`

spécif. du serveur
spécif. du port distant
spécif. du port local

- `InputStream getInputStream()`
- `OutputStream getOutputStream()`
- **synchronized void** `close()`

flux pour la lecture
flux pour l'écriture
ferme le socket

Exemple :

```
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();           // fermeture partielle

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

            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);}
        }
    }
}
```


Côté serveur : la classe `ServerSocket` gère les écoutes

- `ServerSocket (int port)` crée une écoute sur le port
(sans limite de taille de buffer)
- `ServerSocket (int port, int taille)` (avec limite la taille)
- `ServerSocket (int port, int taille, InetAddress adr)`
écoute sur l'interface IP `adr`
- `Socket accept()` début d'attente de clients
- `void close()` fin du serveur

Exemple :

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

public class Main {
    private ServerSocket serverSocket;
    private Socket socket;

    public Main(int port) {
        try { serverSocket = new ServerSocket(port, 1);}
            // creation du serveur
        catch (IOException e) {}           // erreur de création
    }

    public static void main(String[] args) {
        int port;
        try {port = Integer.parseInt(args[0]);}
        catch (Exception e) {port = 0;}    // donc valeur au hasard
        Main ct = new Main(port);
        ct.clientMgr();
    }
    ... // cf page suivante
```

```

...
public void clientMgr() {
    while (true) {                                     // écoute
        try { Socket socket = serverSocket.accept();
            Thread inputThread =
                new InputThread(socket.getInputStream());
            inputThread.start();                        // thread pour lecture

            Thread outputThread =
                new OutputThread(socket.getOutputStream());
            outputThread.start();                       // thread pour écriture

            try { inputThread.join();
                outputThread.join(); } //attente de fin R/W
            catch (InterruptedException e) { }          // interruption de thread
        }
        catch (IOException e) {System.out.println(e);}
        finally {
            try { if (socket != null) socket.close(); }
            catch (IOException e) {}
        }
    }
}

```

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

class InputThread extends Thread {
    InputStreamReader in;

    public InputThread(InputStream in) {
        this.in = new InputStreamReader(in);
    }

    public void run() {
        try {
            int i;
            while ((i = in.read()) != -1) { System.out.write(i); }
        }
        catch (SocketException e) {} // cas socket fermé
        catch (IOException e) {}
        try { in.close(); }
        catch (IOException e) {}
    }
}
```

```
import java.io.*;

class OutputThread extends Thread {
    OutputStreamWriter out;

    public OutputThread(OutputStream out) {
        this.out = new OutputStreamWriter(out);
    }

    public void run() {
        String ligne;
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        try {
            while (true) {
                ligne = in.readLine();
                if (ligne.equals(".")) break;
                out.write(ligne + "\r\n");
                out.flush();
            }
        } catch (IOException e) {}
        try { out.close(); }
        catch (IOException e) {}
    }
}
```

Connexions TCP sécurisés

Packages : (dans le JDK standard à partir de la version 1.4)

- `javax.net.ssl` classes abstraites : communication sécurisée
- `javax.net` sockets sécurisés
- `java.security.cert` gestion de clés pour SSL
- `com.sun.net.ssl` algo de cryptage (==> provider)

En fait intègre le cryptage dans l'envoi de données et non au niveau du flux de données.

Le "cryptage" consiste à spécifier

- si il y a authentification (méthode RSA)
- si il y a cryptage des blocs (DES ou RC4)
- si il y a contrôle de somme (i.e. signature) (MD5 ou SHA)

Grandes lignes de la méthode :

- ❶ spécification du provider de l'algo de cryptage
 - soit ligne de spécif. dans le fichier `java.security`
 - soit méthode `addProvider(..)` de la classe `Security`
- ❷ création de la "fabrique" de cryptage via la classe `SSLSocketFactory`
méthode `getDefault()` existe
- ❸ création de socket par la méthode d'instance `createSocket(...)` de la classe `SSLSocketFactory`

Méthode :

- 1 générer les clés publiques et certificats (commande `keytool`)
- 2 authentifier les certificats (==> tiers)
- 3 créer un `SSLContext` pour l'algo de cryptage
- 4 créer un `KeyManagerFactory` pour le gestionnaire de clés
- 5 créer un `KeyStore` pour la base de clés et certificats
- 6 ... et l'initialiser avec le fichier de clés
- 7 initialiser le `KeyManagerFactory` avec le résultat de 6.
- 8 initialiser le `SSLContext` avec le résultat du 7.
- 9 créer un `SSLServerSocketFactory` pour la génération de serveurs
- 10 créer un `SSLServerSocket` à partir du résultat de 9.
- 11 créer une socket en acceptant les connexions sur le `SSLServerSocket`

possibilité de préciser les protocoles de cryptage disponibles :

- **public abstract** `String[] getEnabledCypherSuites()`
retourne la liste des méthodes de cryptage (une chaîne de caractères = une méthode)
- **public abstract void** `setEnabledCypherSuites(String[] c)`
spécifie les codes (à partir des codes `c` disponibles dans l'implémentation)

possibilité d'avoir un jeu de clés par session

(par défaut un jeu de clés est réutilisé entre deux connexions)

Exemples d'opérations à effectuer :

● SERVEUR

```
% keytool -genkey -keystore Fichier_Certif
// génération d'un fichier de clé
// (en fait d'un certificat,
//      d'une clé privée et d'une clé publique)

% keytool -list -keystore Fichier_Certif
// ne sert qu'à voir le contenu du fichier de clés

% keytool -selfcert -keystore Fichier_Certif
// auto-certification de la clé publique du serveur
//      (dans Fichier_Certif)

% java -Djavax.net.debug=ssl:handshake:verbose \
      ServerMaitre 20000 certif
// certif : mot de passe permettant d'entrer
//      dans le fichier des clés
// version manuelle

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

● CLIENT

```
% cp Fichier_Certif cacerts
// fichier "client" des certificats
// exportation uniquement des certificats: keytool -export

% java -Djavax.net.ssl.trustStore=cacerts \
      -Djavax.net.debug=ssl:handshake:verbose \
      Client localhost 20000 Client.java
// cacerts : nom du fichier de certificat du client

// -Dxxx : spécification d'une valeur d'attribut
// dans l'environnement de la machine virtuelle
```

Exemple de client de Socket SSL :

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

public class ClientHTTPS {

    private final int portHTTPS = 443;    // port https par défaut
    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 {                                     // envoi de données
        Writer output =
            new OutputStreamWriter(sslSocket.getOutputStream());
        output.write("GET_https://" + host + "/_HTTP_1.1\r\n\r\n");
        output.flush();

                                                // lecture de la réponse
        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);}
}
}

```

Exemple de partie de code serveur avec Socket SSL :

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

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

// 1) spécif de la fabrique de cryptage
KeyManagerFactory kmf=null;
SSLContext context=null;
try {
    context = SSLContext.getInstance("SSLv3");
    kmf = KeyManagerFactory.getInstance("SunX509");
} catch (NoSuchAlgorithmException e1) { e1.printStackTrace(); }

// 2) spécif du gestionnaire de clés
KeyStore ks = null;
try {
    ks = KeyStore.getInstance("JKS");
} catch (KeyStoreException e2) { e2.printStackTrace(); }
... // cf page suivante
```

Exemple de partie de code serveur avec Socket SSL :

```
...  
// 3) récupération du certificat (et de la clé)  
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) paramétrage de la fabrique de cryptage par la clé  
try {  
    kmf.init(ks, passPhrase);  
} catch (KeyStoreException e4) { e4.printStackTrace();  
} catch (NoSuchAlgorithmException e4) { e4.printStackTrace();  
} catch (UnrecoverableKeyException e4) { e4.printStackTrace();  
}  
... // cf page suivante
```



```

...
// 5) spécification du contexte de génération de SSLServerSocket
try {
    context.init(kmf.getKeyManagers(), null, null);
} catch (KeyManagementException e5) { e5.printStackTrace();
}

// 6) création de la fabrique de SSLServerSocket
SSLServerSocketFactory factory =
    context.getServerSocketFactory ();

// 7) création d'un 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);
}

}

...

```

Il y a aussi la possibilité de spécifier des droits particuliers sur la connexion à un serveur (ou la connexion d'une machine vers un serveur) :

- `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");`

2 méthodes de spécification :

- un fichier `java.policy` spécifiant ces droits (fichier de configuration `$JDKHOME/jre/lib/security/java.policy` chargé lors du lancement de la machine virtuelle Java)
- une classe redéfinissant la politique de sécurité

Connexions UDP entre machines

- intérêt : rapide
- défaut : absolument pas sûr
- à n'utiliser que pour les connexions sans session et pour de "petits" paquets
- données gérées par la classe `DatagramPacket` avec les méthodes `receive()` et `send()`

Exemple de serveur :

```
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) {}
    }
}
```

Exemple de réception de paquets (extrait de programme) :

```
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) {}
        }
    }
}
```

Connexions entre machines en multicast

Dans le cas de communications partagées par de nombreux utilisateurs (vidéo, newsgroups, ...), le mécanisme point à point est fastidieux. Le mécanisme multipoint permet de remédier à cela de la manière suivante :

- un paquet de données est envoyé au groupe entier. Les routeurs du réseau Internet sont chargés de la distribution (autant que possible sans duplication inutile).
- chaque groupe d'utilisateurs est référencé par une adresse IP spécifique.
- le protocole UDP est utilisé
- les paquets sont envoyés sur une "zone géographique"

Les mécanismes sont les mêmes que pour les sockets avec datagrammes UDP. Il suffit juste de "se joindre" au groupe (et de le quitter en fin de session) :

- `MulticastSocket (int port)` crée un socket multipoint
- `MulticastSocket joinGroup (InetAddress ia)`
méthode permettant de joindre le socket au groupe référencé par l'IP `ia`
- `MulticastSocket leaveGroup (InetAddress ia)`
méthode permettant de se séparer du groupe
- `MulticastSocket receive (DatagramPacket dp)`
reçoit un paquet UDP
- `MulticastSocket send (DatagramPacket dp, byte ttl)`
envoie un paquet (le `ttl` précise le nombre de routeurs que ce paquet pourra "traverser")

3 Réseau : adressage et sockets

- Connexions réseau
- URL
- RMI et RMI-IIOP

Classes `URL` et `URLConnection`

- `URL` : classe gérant l'objet `URL` (i.e. les infos statiques liées à cet `URL`)
- `URLConnection` : classe gérant les connexions sur une `URL` (i.e. la gestion du socket permettant la lecture de données)

(exception : MalformedURLException)

```
public URL(String url)
    // new URL("http://www.univ-paris13.fr/index.html");
    // exception si le protocole n'est pas géré par la machine

public URL( String protocole, // http
    String machine, // www.univ-paris13.fr
    String fichier) // index.html

public URL( String protocole,
    String machine,
    int port, // 80 (par défaut pour http)
    String fichier)

public URL( URL base,
    String relative)

public URL( String protocole,
    String machine,
    int port,
    String fichier,
    URLStreamHandler handler)
    // permet de spécifier le gestionnaire de protocoles à utiliser
```

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

```
int getPort()
```

Entrées / Sorties :

`URLConnection openConnection()`

ouvre une connexion sur l'URL,
retourne un socket sur cet URL
(donc possibilité d'effectuer entrées et sorties)

`InputStream openStream()`

ouvre une connexion sur l'URL,
répond à l'authentification si nécessaire,
instancie un `InputStream` pour récupérer les données
(lecture standard)

`Object getContent()`

récupère les données d'une URL,
puis les sort dans un format (i.e. une instance d'objet)
"normalement" défini par le protocole.

`getClass()` sur l'objet permet de savoir à quel
type de données on a affaire

(`URLImageSource` pour gif, `MeteredStream` pour applet, ...)

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

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

Une fois la connexion établie sur un URL (quelque soit son type), on peut :

- récupérer les spécifications de l'url
- configurer la connexion
- envoyer / recevoir des données

Spécifications de l'url :

<code>String getContentType()</code>	type MIME (text/html, image/gif)
<code>int getContentLength()</code>	
<code>String getContentEncoding()</code>	(null si pas d'encodage, sinon x-gzip, ...)
<code>long getDate()</code>	
<code>getExpiration()</code>	
<code>getLastModified()</code>	
<code>getHeaderField(String header)</code>	("content-type", ...)

Configuration de la connexion pour

- gestion de cache,
- autorisation de mot de passe,
- spécif de l'en-tête envoyé lors de requêtes au serveur

Réception / envoi de données

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

Il existe des sous-classes spécifiques pour :

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

Exemple :

```
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 connection = urlMail.openConnection();
            PrintStream p = new PrintStream(connection.getOutputStream());
            p.println("Subject:_test\r\n\r\n_corps_du_mail");
            // un message est constitué d'un en-tête
            // séparé d'une ligne vide avant le corps du message
            p.close();
        } catch (IOException e) {System.out.println(e);}
    } }
```

%java Mailer

ou

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

si System.setProperty non mis, où mail.host est une propriété permettant de préciser l'adresse du serveur SMTP.

Gestion de protocoles et communication

Il est possible de gérer "à la main" les protocoles lorsque ceux-ci ne sont pas ou incorrectement traités par la machine virtuelle.

Implantation standard de la classe 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);
    }
    ... // cf pages suivante
```



```

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;
        }
    }
}

```

Exemple complet (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 :

```
5501000101010001000101010001
```

Exécution :

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

3 Réseau : adressage et sockets

- Connexions réseau
- URL
- RMI et RMI-IIOP

RMI :

- Principe développé par Sun permettant d'appeler des méthodes exposées sur une autre machine virtuelle
- Mécanisme propre à Java
- Utilise la bibliothèque `java.rmi`

RMI-IIOP :

- IIOP : *Internet Inter-Orb Protocol*
- Compatibilité avec CORBA (donc interopérable avec d'autres environnements d'appels de fonctions ou de méthodes à distance)
- Obligatoire avec l'environnement EJB (*Enterprise Java Beans*)
- Utilise les bibliothèques `java.rmi` et `javax.rmi`

Principales différences :

- RMI : chargement de classes et activation d'objets à chaud à distance
- donc classes distinctes pour identifier les objets utilisables à distance :
 - RMI : `java.rmi.server.RemoteObject`
 - RMI-IIOP : `javax.rmi.PortableRemoteObject`

3 machines virtuelles :

- A : va contenir le ou les objets dont d'autres machines virtuelles utiliseront les méthodes
- B : machine virtuelle appelant les méthodes des objets exposés par A
- C : machine virtuelle jouant le rôle de serveur en exposant les objets utilisables à distance par des noms symboliques (comme un DNS)

Principe :

- Lancement de C : serveur écoutant sur le port 1099 les requêtes de déclaration d'objets ou de recherche d'objets (la commande `rmiregistry` lance un tel serveur)
- Lancement de A :
 - création des objets que A veut présenter
 - ouverture d'un port en attente de requête vers les objets "publics"
 - requête à C pour rendre "publics" les objets (numéro de port sur A, adresse IP de A, ...)
- Lancement de B :
 - requête à C pour récupérer les modalités d'utilisation de l'objet sur A
 - exécution de méthodes

La face cachée : Nécessite la création de classes spéciales (donc des fichiers) permettant de sérialiser et désérialiser les requêtes et les structures des classes des objets exportés,

- une interface permet de spécifier la structure de la classe de l'objet exporté. Cette interface doit exister sur A et B.
- Côté exportateur A : un fichier squelette (*skeleton*)
- Côté importateur B : un fichier souche (*stub*)
- Dans les versions initiales de java, 2 fichiers distincts, maintenant un seul fichier stub
- Ce fichier stub est généré par la commande `rmic` sur A
- Le stub doit être copié sur B (et gardé sur A!).

Exemple (étape 1) :

Interface à copier pour A et B : classe `Display.java`

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

public interface Display extends Remote{
    //extension de l'interface remote
    public void showDigest(File fileName) throws RemoteException;
}
```

Exemple (étape 2) :

Implantation de l'interface pour A : classe `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() + ":\n";
        if (digest != null) {
            for (int i = 0; i< digest.length; i++) {result += digest[i] + "\n";}
        } else { result += "unusable_digest";};
        return result;
    }
}
```

Exemple (étape 3) :

Compiler cette implantation et récupérer un stub qui servira aux clients et un skeleton à laisser sur le serveur

```
% javac Display.java // ==> Display.class
% javac DisplayClass.java // ==> DisplayClass.class
% rmic DisplayClass // ==> DisplayClass_Stub.class et DisplayClass_Skel.class
% cp ... // copie de DisplayClass_Stub.class sur la machine cliente
```

Exemple (étape 4) :

Ecrire un code `DisplayPublisher.java` permettant la publication sur le serveur RMI C (ce code peut faire partie du code "ordinaire" de A)

```
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);
        // crée le thread en attente d'appel sur l'objet (ici display),
        // et associe un port TCP pour les connexions

        Naming.rebind("/UnDisplayDistant", display);
        // la classe Naming cree les liens entre URL et objet
        // URL :: rmi://host:port/nom
        // objet implante Remote
        // bind : le nom ne doit pas exister
        // rebind : pas de restriction
        // unbind : suppression du lien
    }
}
```

Exemple (étape 5) :

Exporter l'objet vers le serveur RMI

```
% rmiregistry &  
    // Création d'un démon serveur de noms si celui-ci n'est pas déjà lancé  
    // par défaut en attente sur le port 1099  
    // Autre possibilité : utiliser la classe LocateRegistry  
    // qui permet de créer des serveurs de noms  
% javac DisplayPublisher.java  
% java DisplayPublisher
```

La dernière commande *crée* la machine virtuelle A, donc reste en exécution.

Exemple (étape 6) :

Code `DisplayClient.java` à faire tourner sur 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);
    }
}
```

Remarques :

- La classe `HelloFrame` doit implanter l'interface `Serializable` (elle est par ailleurs écrite de manière totalement standard).
- Plusieurs appels distants peuvent avoir lieu en même temps sur le même objet (sur le serveur) : chaque exécution distante de méthode est faite sur un thread. Il faut donc veiller à respecter les principes de gestion de concurrence sur les variables de l'objet.
- Les paramètres passés en argument sont copiés de la machine cliente vers la machine serveur : les paramètres de type non primitif ne seront pas modifiés par l'exécution distante.

Le mécanisme RMI utilise implicitement le chargement de classes à chaud, effectuable en général :

- `java.lang.ClassLoader` : classe abstraite générique
- `getSystemClassLoader()` : chargeur standard (i.e. local)

Exemple :

```
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));  
    }  
}
```

Sous-classes avec chargeur spécifique :

- `security.SecureClassLoader` : chargeur avec test de sécurité
- `java.net.URLClassLoader` : chargeur de classes distantes
- `rmi.server.RMIClassLoader` : spécifique à RMI

Exemple : (on suppose que `Test.class` se trouve dans le répertoire `/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));

    }
}
```

Exemple sur 3 machines distinctes :

- 1 machine **client**
- 1 machine **httphost** sur laquelle tourne un serveur HTTP
- 1 machine **codehost** sur laquelle existe une machine virtuelle et un objet publié connu par **httphost** mais non public

L'objectif est que **client** exécute un code téléchargé de **httphost** qui va lui-même exécuter une méthode d'un objet de **codehost**.

Sur la machine **client** :

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

public class GenericClient {

    public static void main (String[] args) {
        try {
            System.setProperty( "java.security.policy", "maPolicy");
            System.setSecurityManager(new SecurityManager());

            Class c = RMIClassLoader.loadClass( "http://httpost:20000/", "SpecialClient");
            Runnable client = (Runnable) c.newInstance();
            client.run();
        } catch (Exception e) {System.out.println(e); }
    } }
```

```
public interface Hello extends java.rmi.Remote {
    public String hello(String s) throws java.rmi.RemoteException;
}
```

```
// fichier maPolicy
grant {
    permission java.security.AllPermission;
};
```

Sur httphost : le fichier Hello.java et

```
import java.rmi.*; import java.net.*;

public class SpecialClient implements Runnable {

    public void run() {
        try {
            Remote r = Naming.lookup("//codehost/Hello");
            System.out.println((
                (Hello) r).hello(InetAddress.getLocalHost().getHostAddress()));
        } catch (Exception e) { System.out.println(e); }
    }
}
```

Sur **codehost** : le fichier `Hello.java` et

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

public class ImpHello extends UnicastRemoteObject implements Hello {
    public ImpHello() throws RemoteException {
        super(10000);
    }

    public String hello(String s) { return s; }

    public static void main(String[] args) {
        try {
            System.setProperty( "java.rmi.server.codebase","http://httphost:20000/");
            Remote r = new ImpHello();
            Naming.rebind("/Hello",r);
        } catch (Exception e) { System.out.println(e); }
    } }
```

Il faut lancer `rmiregistry` sur les machines **client** et **codehost**.
Puis lancer sur la machine **codehost** le programme
`SpecialClient` (qui boucle en attendant des requêtes de
clients).
Enfin le programme `GenericClient` sur la machine **client**.

4 Gestion d'annotations

Une **annotation** est une *méta-donnée* permettant à l’“environnement”

- de générer des fichiers de configuration (pour le déploiement, la compilation finale)
- de générer des interfaces (en précisant les méthodes à y inclure), des classes subsidiaires, ...
- de lier des noms de méthodes ou de spécifier des noms externes (base de donnée, ...)
- de déterminer des informations annexes (documentation, tests,...)

- Une annotation est inscrite dans le code java même en commençant par le symbole @.
- L'environnement doit comprendre des programmes permettant d'interpréter les annotations.
- L'interprétation des annotations peut avoir lieu sur le fichier `.java` ou sur le fichier `.class` si l'annotation y est conservée (i.e. la méta-annotation `@Retention` est utilisée dans le fichier source).
- Les annotations sont disponibles depuis la version 5 de Java.

- Une annotation se pose avant n'importe quel modifieur (p.e. `public`) d'une classe, d'une méthode ...
- Elle peut être combinée en une séquence d'annotations
- Une annotation peut être annotée une autre annotation
- une annotation est *posée* dans un programme java :

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

OU

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

OU

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

(si l'annotation n'a qu'un attribut de nom `String value()`)

- une annotation est *définie* dans un programme Java comme un ensemble éventuellement vide d'attributs :

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

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

- une annotation est *utilisée* par un programme :

```
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
            }
        }
    }
}
```

Pour l'exemple précédent, il faut que l'annotation ait été conservée dans le fichier `.class`, d'où l'utilisation de l'annotation `Retention` (cf plus loin)

Pour chaque type d'annotation `javac` recherche dans son environnement (`classpath`) une classe dont le nom est celle du type d'annotation. Il ajoute alors au code compilé le contenu de l'annotation (valeurs par défaut, ...).

Plus de 60 annotations en standard dont :

- `@Deprecated` : (devant une méthode) indique que la méthode qui suit n'est pas recommandée (i.e. son usage génèrera un warning)
- `@Override` : (devant une méthode) indique que la méthode qui suit *doit* surcharger une méthode d'une super-classe
- `@SuppressWarnings(type)` : supprime les warnings pour le type donné ("deprecation", "all",...)

Exemple : annotation standard sur un code java

```
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();
    }
}
```

Sans suppression des commentaires :

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

Avec @Override décommenté :

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

Avec @SuppressWarnings décommenté :

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

Les méta-annotations sont en particulier utiles pour définir des annotations. Par exemple :

- `@Retention(type)` : méta-annotation qui conserve selon le `type` (`RetentionPolicy.RUNTIME` dans le code et après chargement dans la VM, `RetentionPolicy.CODE` dans le code seulement) l'annotation utilisée dans un programme.
- `@Target(type)` : méta-annotation qui précise sur quoi peut être appliquée l'annotation (`ElementType.METHOD` pour une méthode, `ElementType.TYPE` pour une classe, une interface, ...).

Exemple : Traitement manuel à l'exécution (1)

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

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

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

```
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
$
```

Exemple : Traitement manuel à l'exécution (1)

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

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

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

**l'annotation Audit sera
conservée à l'exécution**

```
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
$
```

Exemple : Traitement manuel à l'exécution (1)

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

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

déclaration de l'annotation

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

```
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
    }
}
```

appel à l'exécution, traitement
selon la valeur de l'annotation
(cf slide suivant)

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

Exemple : Traitement manuel à l'exécution (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;

            // Récupérer la méthode appelée.
            Method method = object.getClass().getMethod(methodName, paramTypesArray);

            // S'il n'y a pas d'annotation, autoriser l'appel
            if( !method.isAnnotationPresent(Audit.class) ) return;

            // Récupérer la donnée de l'annotation, s'il faut auditer alors le faire
            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); }
}
```

Exemple : Traitement manuel à l'exécution (2)

nombre variable d'arguments
(varargs, Java 5)

```
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;

            // Récupérer la méthode appelée.
            Method method = object.getClass().getMethod(methodName, paramTypesArray);

            // S'il n'y a pas d'annotation, autoriser l'appel
            if( !method.isAnnotationPresent(Audit.class) ) return;

            // Récupérer la donnée de l'annotation, si elle existe sinon alors le faire
            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); }
}
```

utilisation de la réflexivité
dans Java

Exemple : Traitement manuel à l'exécution (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;

            // Récupérer la méthode appelée
            Method method = object.getClass().getMethod(methodName, paramTypesArray);

            // S'il n'y a pas d'annotation, autoriser l'appel
            if( !method.isAnnotationPresent(Audit.class) ) return;

            // Récupérer la donnée de l'annotation, s'il faut auditer alors le faire
            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 test de sa valeur (System.err.println(msg); )
}
```

test de présence de l'annotation associée au code de la méthode (rétention de la déclaration dans le code source)

test de sa valeur

Les annotations sont principalement gérées en phase de compilation :

- En Java 1.5 : l'outil `apt` (*annotation processing tool*) sert au traitement des annotations. Etant séparé du compilateur, la gestion est complexe.
- En Java 6 : `javac` intègre le traitement des annotations par un double mécanisme :
 - une structure liée aux constructions du langage
 - une structure liée à l'arbre de syntaxe abstraite (ASP)

- En Java 6, double mécanisme,
 - (`javax.lang.model.element`) l'interface `Element` et ses sous-interfaces caractérisent un constructeur du langage (paquet, type, classe, méthode). Permet de connaître les sous- et sur-éléments constructeurs. Ne permet pas a priori de connaître le contenu opérationnel.
 - (p.e. `com.sun.source.tree`) l'interface `Tree` et ses sous-interfaces caractérisent l'ASP de Java vu par le compilateur. Permet de connaître les sous-et sur-éléments constructeurs, structures du langage (boucle, ...), et d'accéder au bloc du fichier source associé.
 - Dans les 2 cas, possibilité d'utiliser le pattern *visitor* (méthode `accept(v)` où `v` est un visiteur (implantation de `ElementVisitor` ou `TreeVisitor`) implantant des méthodes `visitXXX()`)

Exemple : Traitement via le compilateur (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 {  
    // true: les accès à cette méthode  
    // sont écrits dans un fichier d'audit  
    boolean value() default false;  
}
```

```
public class Application {  
    @Audit(true)  
    public void methA(String s) {  
        int i = 0;  
        // code de l'application  
    }  
  
    @Audit(false)  
    public void methB(String s) {  
        // code de l'application  
    }  
  
    @Deprecated  
    public void methC() {  
        // code de l'application  
    }  
}
```

Exemple : Traitement via le compilateur (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 et Deprecated
gardées dans le code

Exemple : traitement via le compilateur (2)

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

le code associé à l'annotation
Deprecated génère un warning

```
%javac -processor GestionAudit Application.java
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: {
}
```

Exemple : Traitement via le compilateur (2)

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

jar contenant les classes
permettant le preprocessing
(analyse d'ASP, ...)

```
%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 associée = true
METHOD
Method methA: null
Body: {
    int i = 0;
}
methB(java.lang.String):
Valeur de l'annotation associée = false
METHOD
Method methB: null
Body: {
}
```

le code associé à l'annotation
Audit est exécuté

Exemple : Traitement via le compilateur (3)

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

@SupportedAnnotationTypes({ "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;
    }
}
```

Exemple : Traitement via le compilateur (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;

    @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;
    }
}
```

Liste des annotations gérées par cette classe-processeur

Spécification de la release de Java nécessaire

Exemple : Traitement via le compilateur (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 = "
                               + (Audit.class).value());

            System.out.println(auditVisitor.visit(element, annotations));
        }
        return false;
    }
}
```

classe-processeur par défaut

instanciation (p.e. par javac) avec l'environnement de compilation

processing sur l'environnement courant pour les annotations en 1er argument, si return true alors annotations non analysées par les processeurs suivants

Exemple : Traitement via le compilateur (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;}
    ...
}
```


Exemple : Traitement via le compilateur (4)

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

Classe parcourant l'AST

Type de retour de la méthode visit

Type de l'argument de la méthode visit

Exemple : Traitement via le compilateur (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; }
    ...
}
```

méthode standard de visite d'un élément

Objet noeud de l'AST (ici méthode) permettant l'accès au corps, au nom, aux paramètres, ...

Dernier exemple : JAXB et traitement XML

- JAXB est une API permettant le traitement Java de/vers XML
- la commande `xjc` permet de générer des classes Java à partir d'un schéma XML, et contenant des annotations pour le traitement XML (la commande utilise un *Java Binding Compiler*, `xjc`)
- la classe `JAXBContext` (de `javax.xml.bind`) intègre un analyseur/générateur XML qui fonctionne en utilisant les annotations des classes considérées
- Opérations disponibles :
 - analyse (*unmarshal*) de documents XML en générant un objet Java (équivalent au principe DOM)
 - validation de l'objet Java étant donné un schéma XML
 - génération d'un fichier XML à partir d'un objet

Exemple : Données de formation d'enseignement (1)

formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="formation" type="typeFormation"/>
  <xsd:complexType name="typeFormation">
    <xsd:attribute name="intitule" type="xsd:string" use="required"/>
    <xsd:sequence>
      <xsd:element ref="responsable" /> <xsd:element ref="etudiant"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="etudiant" type="typeEtudiant"/>
  <xsd:complexType name="typeEtudiant">
    <xsd:attribute name="prenom" type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="responsable" type="typeResponsable"/>
  <xsd:complexType name="typeResponsable">
    <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<formation intitule="M2PLS">
  <responsable nom="Fouquere"/>
  <etudiant prenom="Jean" nom="Dupond"/><etudiant prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Exemple : Données de formation d'enseignement (1)

formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="formation" type="typeFormation"/>
  <xsd:complexType name="typeFormation">
    <xsd:attribute name="intitule" type="xsd:string" use="required"/>
    <xsd:sequence>
      <xsd:element ref="responsable" /> <xsd:element ref="etudiant"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="etudiant" type="typeEtudiant"/>
  <xsd:complexType name="typeEtudiant" un élément XML>
    <xsd:attribute name="prenom" type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="responsable" type="typeResponsable"/>
  <xsd:complexType name="typeResponsable">
    <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<formation intitule="M2PLS">
  <responsable nom="Fouquere"/>
  <etudiant prenom="Jean" nom="Dupond"/><etudiant prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Exemple : Données de formation d'enseignement (1)

formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="formation" type="typeFormation"/>
  <xsd:complexType name="typeFormation">
    <xsd:attribute name="intitule" type="xsd:string" use="required"/>
    <xsd:sequence>
      <xsd:element ref="responsable" /> <xsd:element ref="etudiant"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="etudiant" type="typeEtudiant"/>
  <xsd:complexType name="typeEtudiant">
    <xsd:attribute name="prenom" type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="responsable" type="typeResponsable"/>
  <xsd:complexType name="typeResponsable">
    <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

Annotations dans l'image :

- une formation : (pointe vers l'élément `formation`)
- a un intitulé (attribut obligatoire) (pointe vers l'attribut `intitule`)

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<formation intitule="M2PLS">
  <responsable nom="Fouquere"/>
  <etudiant prenom="Jean" nom="Dupond"/><etudiant prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Exemple : Données de formation d'enseignement (1)

formation.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="formation" type="typeFormation"/>
  <xsd:complexType name="typeFormation">
    <xsd:attribute name="intitule" type="xsd:string" use="required"/>
    <xsd:sequence>
      <xsd:element ref="responsable" /> <xsd:element ref="etudiant"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="etudiant" type="typeEtudiant"/>
  <xsd:complexType name="typeEtudiant">
    <xsd:attribute name="prenom" type="xsd:string" use="required"/>
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:element name="responsable" type="typeResponsable"/>
  <xsd:complexType name="typeResponsable">
    <xsd:attribute name="nom" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

Diagram annotations for formation.xsd:

- A blue box labeled "une formation :" points to the `typeFormation` complex type.
- A green box labeled "inclut un responsable" points to the `responsable` element reference.
- A green box labeled "inclut des étudiants" points to the `etudiant` element reference.

M2PLS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<formation intitule="M2PLS">
  <responsable nom="Fouquere"/>
  <etudiant prenom="Jean" nom="Dupond"/><etudiant prenom="Sergei" nom="Sergeievitch"/>
</formation>
```

Diagram annotations for M2PLS.xml:

- A green arrow points from the `responsable` element in the XML to the `responsable` element in the schema.
- A green arrow points from the `etudiant` element in the XML to the `etudiant` element in the schema.

Exemple : Données de formation d'enseignement (2)

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

```
import java.io.*;      import javax.xml.bind.*;      import up13.formation.*;  
public class Test {  
    public static void main(String args[]) {  
        try {  
            JAXBContext jaxbContext = JAXBContext.newInstance ("up13.formation");  
            Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();  
            // pour effectuer une validation XML  
            unmarshaller.setEventHandler(new FormationValidationEventHandler());  
            JAXBElement<TypeFormation> uneFormationJAXB = (JAXBElement<TypeFormation>)  
                unmarshaller.unmarshal(new File("M2PLS.xml"));  
            TypeFormation uneFormation = uneFormationJAXB.getValue();  
  
            System.out.println("Intitulé_:" + uneFormation.getIntitule());  
            TypeResponsable leResponsable = uneFormation.getResponsable();  
            System.out.println("Responsable_:" + leResponsable.getNom());  
  
            // création de données  
            TypeFormation uneAutreFormation = new TypeFormation();  
            uneAutreFormation.setIntitule("M2EID");  
            // Génération XML  
            JAXBElement<TypeFormation> uneAFJAXB = (new up13.formation.ObjectFactory()).  
                createFormation(uneAutreFormation);  
            Marshaller marshaller = jaxbContext.createMarshaller();  
            marshaller.marshal( uneAFJAXB, System.out );  
        } catch (JAXBException e) {System.err.println(e);}  
    } }
```


Exemple : Données de formation d'enseignement (2)

```
%xjc formation.xsd -p "up13.formation" -d .  
// dans up13\formation/, javac *.java  
%javac -cp .;%CLASSPATH *.java
```

crée pour ce schéma XML (.xsd) dans ce paquetage (-p) mis dans ce répertoire (-d)

un fichier ObjectFactory.java (intermédiaire objets JAXB - données) et un fichier-classe par type décrit dans le schéma (cf p.e. slide suivant)

```
import javax.xml.bind.*;  
import up13.formation.*;  
public class Test {  
    public static void main(String args[]) {  
        JAXBContext context = JAXBContext.newInstance ("up13.formation");  
        Marshaller marshaller = context.createMarshaller();  
        Validation XML  
        EventHandler handler = new FormationValidationEventHandler();  
        Formation uneFormationJAXB = (JAXBElement<TypeFormation>)  
            context.unmarshal(new File("M2PLS.xml"));  
        TypeFormation uneFormation = uneFormationJAXB.getValue();  
  
        System.out.println("Intitulé_:" + uneFormation.getIntitule());  
        TypeResponsable leResponsable = uneFormation.getResponsable();  
        System.out.println("Responsable_:" + leResponsable.getNom());  
  
        // création de données  
        TypeFormation uneAutreFormation = new TypeFormation();  
        uneAutreFormation.setIntitule("M2EID");  
  
        // Génération XML  
        JAXBElement<TypeFormation> uneAFJAXB = (new up13.formation.ObjectFactory()).  
            createFormation(uneAutreFormation);  
        Marshaller marshaller = context.createMarshaller();  
        marshaller.marshal(uneAFJAXB, System.out);  
    } catch (JAXBException e) {System.err.println(e);}  
}
```

Exemple : Données de formation d'enseignement (2)

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

```
import java.io.*;      import javax.xml.bind.*;      import up13.formation.*;  
public class Test {  
    public static void main(String args[]) {  
        try {  
            JAXBContext jaxbContext = JAXBContext.newInstance("up13.formation");  
            Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();  
            // pour effectuer une validation XML  
            unmarshaller.setEventHandler(new FormationValidationEventHandler());  
            JAXBElement<TypeFormation> uneFormationJAXB = (JAXBElement<TypeFormation>)  
                unmarshaller.unmarshal(new File("M2PLS.xml"));  
            TypeFormation uneFormation = uneFormationJAXB.getValue();  
  
            System.out.println("analyse d'un document XML on.getIntitule());  
            TypeResponsable leResponsible = uneFormation.getResponsable();  
            System.out.println("et constitution d'in objet getResponsable();  
                                JAXB (vue) dans la VM nsable.getNom());  
  
            // création de données  
            TypeFormation uneAutreFormation = new TypeFormation();  
            uneAutreFormation.setIntitule("M2EID");  
  
            // Génération XML  
            JAXBElement<TypeFormation> uneAFJAXB = (new up13.formation.ObjectFactory()).  
                createFormation(uneAutreFormation);  
            Marshaller marshaller = jaxbContext.createMarshaller();  
            marshaller.marshal(uneAFJAXB, System.out);  
        } catch (JAXBException e) {System.err.println(e);}  
    }  
}
```

Objet contexte permettant la création de :

- un objet analyseur de documents XML

analyse d'un document XML
et constitution d'in objet
JAXB (vue) dans la VM

Exemple : Données de formation d'enseignement (2)

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

```
import java.io.*; import javax.xml.bind.*; import up13.formation.*;  
public class Test {  
    public static void main(String args[]) {  
        try {  
            JAXBContext jaxbContext = JAXBContext.newInstance ("up13.formation");  
            Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();  
            // pour effectuer une validation XML  
            unmarshaller.setEventHandler(new FormationValidationEventHandler());  
            JAXBElement<TypeFormation> uneFormationJAXB = (JAXBElement<TypeFormation>)  
                unmarshaller.unmarshal(new File("M2PLS.xml"));  
            TypeFormation uneFormation = uneFormationJAXB.getValue();  
  
            System.out.println("Intitulé : " + uneFormation.getIntitule());  
            TypeResponsable leRes = uneFormation.getResponsable();  
            System.out.println("Responsable : " + leRes.getResponsable().getNom());  
  
            // création d'un objet JAXB à partir d'un type du schéma XML  
            uneAutreFormation = new TypeFormation();  
            uneAutreFormation.setIntitule("M2EID");  
            // envoi d'un objet JAXB vers un flux de sortie  
            JAXBElement<TypeFormation> uneAFJAXB = (new up13.formation.ObjectFactory()).  
                createFormation(uneAutreFormation);  
            Marshaller marshaller = jaxbContext.createMarshaller();  
            marshaller.marshal(uneAFJAXB, System.out);  
        } catch (JAXBException e) {System.err.println(e);}  
    }  
}
```

Objet contexte permettant la création de :

création d'un objet JAXB à partir d'un type du schéma XML

envoi d'un objet JAXB vers un flux de sortie

- un objet générateur de documents XML

Exemple : Données de formation d'enseignement (3)

```
package up13.formation;

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

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "typeFormation", propOrder = {
    "responsable",
    "etudiant"
})
public class TypeFormation {

    @XmlElement(required = true)
    protected TypeResponsable responsable;
    @XmlElement(required = true)
    protected List<TypeEtudiant> etudiant;
    @XmlAttribute(required = true)
    protected String intitule;

    /**
     * Gets the value of the responsable property.
     */
    public TypeResponsable getResponsable() {
        return responsable;
    }
    ...
}
```

Exemple : Données de formation d'enseignement (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("Document_de_définition_de_formation_invalide_:_"
                               + locator.getURL());
            System.out.println("Erreur_:_" + ve.getMessage());
            System.out.println("Colonne_" +
                               locator.getColumnNumber() +
                               ",_ligne_"
                               + locator.getLineNumber());
        }
        return true;
    }
}
```

Exemple : Donner un gestionnaire d'événements pour l'analyseur XML (cas d'erreurs, ...)

```
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("Document_de_définition_de_formation_invalide_:_"
                               + locator.getURL());
            System.out.println("Erreur_:_" + ve.getMessage());
            System.out.println("Colonne_" +
                               locator.getColumnNumber() +
                               ",_ligne_"
                               + locator.getLineNumber());
        }
        return true;
    }
}
```

5 Outils : messagerie, serveur de noms

- Messagerie
- Serveur de noms

5 Outils : messagerie, serveur de noms

- Messagerie
- Serveur de noms

La messagerie permet l'envoi asynchrone d'informations.
Utilisé en messagerie "ordinaire", comme en gestion asynchrone entre processus.

- `javax.mail.*` est la bibliothèque contenant les classes nécessaires (pas dans le JDK standard, mais dans J2EE, cf aussi <http://developers.sun.com/downloads/>).
- Définit une API utilisable de manière générique et une API pour les différents fournisseurs de protocoles-services.
- Les API des protocoles suivants sont donnés en standard :
 - pour l'envoi de messages (protocole SMTP)
 - pour la lecture (protocoles IMAP4 et POP3)
- Base de certains webmails

Principe d'envoi :

- `Session`
 - définit l'environnement d'envoi à partir des propriétés :
 - `mail.transport.protocol` : protocole pour l'envoi (p.e. "smtp")
 - `mail.smtp.host` : machine relais pour le protocole d'envoi choisi
 - `getInstance()` crée une instance de session à partir de propriétés données en argument, et optionnellement d'un objet permettant l'authentification
- `MimeMessage` pour la constitution de message au format Mime et incluant les infos de session
- `Transport` permet l'envoi effectif d'un message donné en argument (intègre aussi un mécanisme d'écouteurs)

Exemple :

```
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");
        // ou -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,          // ou CC ou BCC
            new InternetAddress("une.personne@univ-paris13.fr")
        );
        message.setSubject("Test");
        message.setText("ceci_est_le_contenu_du_test_!");

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

Comment récupérer le serveur de mail de son environnement local (sous linux/unix) :
quand son domaine d'adressage mail est univ-paris13.fr,

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

Principe de réception :

- `Session`
 - définit (aussi !) l'environnement de réception à partir de propriété :
 - `mail.store.protocol` : protocole pour la lecture (p.e. "imap")
 - `getInstance()` crée une instance de session à partir de propriétés données en argument, et optionnellement d'un objet permettant l'authentification
- `Store` correspond à une boîte de réception
- `Folder` correspond à un répertoire dans une boîte
- `Message` correspond à un message dans un répertoire de réception

Exemple :

```
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 Outils : messagerie, serveur de noms

- Messagerie
- Serveur de noms

Les serveurs de noms permettent de commuter entre clé de référencement et propriétés associées :

- Service de système de fichiers : ext3, NTFS, ...
- Service de noms de domaine : DNS
- Service d'annuaires : NIS, LDAP, Active Directory, ...

Un **contexte** est un ensemble de paires clé-valeur (*bindings*) :

- les clés sont organisées en arborescence :
 - `/usr/bin` est un sous-contexte de `/usr` (ext3)
 - `univ-paris13.fr` est un sous-contexte de `fr` (DNS)
 - `ou=structures,dc=univ-rennes1,dc=fr` est un sous-contexte de `dc=univ-rennes1,dc=fr` (LDAP)
- le contexte permet les opérations suivantes :
 - lier une valeur à une clé
 - supprimer la liaison
 - lister les liaisons

L'environnement logiciel nécessaire est donné comme :

- une API `javax.naming.*` pour la programmation dont les méthodes appellent un SPI
- un SPI qui dépend du serveur externe (*Service Provider Interface*)
- l'interface `Context` définit la structure abstraite nécessaire

L'objet initial est défini par :

- `InitialContext` à partir de :
 - un SPI donné comme valeur de la propriété
`naming.factory.initial`
 - une racine

Le choix de la SPI dépend du type du domaine de noms, il s'agit d'une classe qui sera chargée "à chaud".

- `com.sun.jndi.fscontext.RefFSContextFactory` est la SPI donnée avec Java pour les systèmes de gestion de fichiers.

Opérations standard :

- `list()` : liste le contexte racine ou celui donné en argument
- `rename()` : renomme le contexte racine ou celui donné en argument
- `createSubContext()` : crée un sous-contexte
- `lookup()` : change de / recherche un contexte

```
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-serveur : contrôle

Client-serveur

Quelles attaques massives ?

- spam sur un serveur de messagerie
- attaque de type déni de service sur pile serveur
- robot sur formulaire

Comment se prémunir d'attaques massives ? 2 types

- Distinguer humain de machine
- Distinguer “vrai” client de “faux” client

Aucune parade n'est totalement efficace.

Captcha :

(completely automated public Turing test to tell computers and humans apart, 2000, CMU)

Déformation de lettres et chiffres pour une reconnaissance automatique difficile :

- déformation de chaque lettre
- ajout de couleurs
- accollement de lettres
- surlignage
- version sonore



En Java, plusieurs API et paquetages sont disponibles :

- JCaptcha (voir exemple)
- SimpleCaptcha
- reCAPTCHA

Partie d'une 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/>
    </td>
    <td><input type="text" name="inCaptchaChars"/></td>
  </tr>
  <tr>
    <td colspan="2" align="right"><input type="submit" value="Submit"/></td>
  </tr>
</table>
</form>
...
```

avec un mapping `captcha-demos/simpleCaptchaServlet` sur la classe de `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;
    }
}

```

Idée : Faire travailler le client ! (1998)

Protocole : **Proof-of-work**

Côté client :

- recherche d'une chaîne de caractères X telle que le hachage de la chaîne "<date> :<adresse> :X" commence par 20 (ou 50, ...) bits à zéro
- envoi de X et du hachage au serveur

Côté serveur :

- vérification de X avec le hachage envoyés par le client

Mécanisme utilisé pour :

- antispam sur la messagerie (*Hashcash*)
- sécurité des transactions en système distribué (*bitcoin*)

bitcoin : système de transactions à monnaie virtuelle

- une adresse = une clef publique
- la clef privée est le mécanisme cryptographique assurant du détenteur de bitcoins
- réseau pair à pair de machines (accès libre)
- chargement par chaque machine de la base de données de toutes les transactions effectuées
- les demandes de transactions sont rassemblées en bloc
- choix aléatoire de la machine effectuant la preuve de travail sur le bloc (10mn, pour éviter les doubles transactions)

7 Politique de sécurité en Java

- Introduction
- java.security
- java.policy

7 Politique de sécurité en Java

- Introduction
- java.security
- java.policy

(policy security)

Une **politique de sécurité** détermine les permissions, les droits associées au code provenant d'autres sources (que ceux sur la machine virtuelle) et aux connexions de machines clientes.

Classes importantes :

- `java.lang.Policy` : détermine la politique de sécurité
- `java.lang.SecurityManager` : applique la politique de sécurité

Par défaut : pas de gestion de sécurité.

Si un gestion de sécurité est chargé :

- Envoi systématique (API Java) de requête au gestionnaire de sécurité pour savoir si une méthode est potentiellement risquée.
- Si potentiellement risquée alors test au vu de la politique de sécurité.

Exemple :

- `checkRead()` : teste si un thread a le droit de lire dans tel fichier,
- `checkWrite()` : teste si un thread a le droit d'écrire dans tel fichier.

Opérations testées :

- modification ou accès à des propriétés "système"
- modification sur le programme, un thread ou un processus (priorité, fin, ...)
- demande de connexion socket (de host/port ou vers host/port), lancement d'un serveur-socket
- gestion de fichiers (création, destruction, lecture, écriture)
- création d'un chargeur de classes
- chargement dynamique de classes (avec méthodes natives ou de librairies)
- modification de librairies

Ce qui n'est pas testé :

- création de threads (DoS)
- gestion de la mémoire (DoS)
- certaines opérations d'applets : envoi de mail, contrôle des data

2 méthodes pour définir des politiques de sécurité (i.e. des permissions) :

- Dans un code :

```
p1 = new SocketPermission("F204-2.ig-edu.univ-paris13.fr:7777",  
                          "connect");  
p2 = new SocketPermission("localhost:1024-",  
                          "accept,connect,listen");
```

puis définir une sous-classe de la classe `Permission` en intégrant les permissions précédentes.

- Indiquer comment les utiliser en définissant des fichiers de description de politique de sécurité (cf. ensuite)

Répertoire général pour les fichiers de description de politique de sécurité :

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

qui contient :

- `java.security` : fichier principal !!
- `java.policy` : fichier principal pour la politique de sécurité
- `javafx.policy` : politique pour FX
- `javaws.policy` : politique pour Java Web Start
- `trusted.libraries` : librairies acceptées
- `blacklist` : blacklist de signatures
- `cacerts` : certificats acceptés
- `local_policy.jar` : politique pour les tailles de cryptage, acceptable par "tout" pays
- `US_export_policy.jar` : politique pour les tailles de cryptage aux USA

7 Politique de sécurité en Java

- Introduction
- **java.security**
- java.policy

Fichier de listes attribut-valeur + lignes de commentaire

```
#  
# Class to instantiate as the system Policy. This is the name  
# 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
```

Autres informations présentes :

- classes pour les fournisseurs d'algo de cryptage
- restriction sur les certificats
- liste des paquetages à accès contrôlé

7 Politique de sécurité en Java

- Introduction
- java.security
- java.policy

java.policy : fichier par défaut

```
// 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" properties 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" properties 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";
};
```

```
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";  
    ...  
};
```

- `signedBy` : liste d'alias de certificats accessibles dans le `keystore` : autorisation pour un code signé en clé privée correspondant à un des alias
- `codeBase` : spécifie d'où peut venir le code (par défaut : de n'importe où)
- `principal` : spécifie un nom d'utilisateur ou d'organisateur, i.e. une chaîne de caractères
- `permission : permission_class_name` doit être une sous-classe de la classe `Permission`, liste les permissions autorisées.

Pour les permissions : sous-classe de

`java.security.Permission`

- `AllPermission` : accepte tout
- `RuntimePermission` : aspects système (chargeurs, thread, ...)
- `FilePermission` : read, write, execute, delete, readlink
- `MBeanPermission` : gestion de bean-ressource
- `PrivateCredentialPermission` : contrôle d'accès sur les noms (cf. Principal)
- `ServicePermission` : contrôle d'accès sur les autorisations
- `SocketPermission` : contrôle sur les sockets (accept, connect, listen, resolve)
- `UnresolvedPermission`

pour le `codeBase` : spécifie le répertoire où se trouve les `.class` chargeables

- obligatoirement une URL
- forme `...truc` : fichier `truc`
- forme `...truc/` : dans le répertoire `truc` sauf les fichiers `truc/xx.jar`
- forme `...truc/*` : dans le répertoire `truc` avec les fichiers `truc/xx.jar`
- forme `...truc/-` : dans le répertoire `truc` et récursif.

Lancement du gestionnaire de sécurité :

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

ou bien dans le code :

```
System.setSecurityManager (java.lang.SecurityManager)
```

Ajout de fichier de politique de sécurité au lancement :

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

Uniquement le fichier de politique de sécurité mentionné :

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