Environnement et langages évolués

Christophe Fouqueré

Master Informatique 2ème année

christophe.fouquere@univ-paris13.fr
A204, Université de 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
- **XHTML**, I. Graham, Eyrolles, 2001
- **EJB fondamental**, Ed Roman, S. Ambler, T. Jewel, Eyrolles, 2002
- **EJB Patterns**, D. Alur, J. Crupi, D. Malks, Prentice Hall, 2008
- et bien sûr `http://docs.oracle.com`

  en particulier `http://docs.oracle.com/javase/{6,7,8}/docs/api/`

  et `http://docs.oracle.com/javaee/7/api/`

Objectifs du cours :

- Usage d'environnements de développements (Eclipse, Netbeans) : apprentissage en TP
- Compréhension des pratiques logicielles en environnement ouvert
- Cas des sites web et services web : mécanismes sous-jacents et mise en oeuvre pratique
- Quelques frameworks
- Questions liées à la sécurité (typage, ...)

Plan de cours

1. rappels Java
2. EJB/J2EE
3. Hibernate
4. JSF
5. Struts
6. Spring

(Devoir ou) Exposé :

- JUnit
- Maven
- Ant
- Git / Svn
- Ruby on Rails
- framework php
- Dot-net
- C#
- ...

Java : Une classe = structure + méthodes

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

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

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

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

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

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

public class Personne {
    private String nom;
    protec                g> prenoms;
    public                ing societe;
    privat                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;}
    }

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

nom du paquetage (donc test.Personne)

importation de classes

définition de classe (convention: début majuscule)

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;
import java.util.*;

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

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

utilisable localement

utilisable par héritage

utilisable globalement

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

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

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

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

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

non sérialisable

non modifiable après instanciation

variable de classe

test d'instance

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

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

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

public class Personne {
    private String nom;
    protected List<String> prenoms;
    public transient String societe;
    private static final String societeException
                                                    nconnu");
    public boolean defini;
    public void identifiant()
                                    ("je m'appelle" + nom);}
    Personne(Object o) {
        if (o                                   .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)
```

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

```
import java.io.*;

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

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[] diges
        ...
    }
    ...
}
```

classe implantant les méthodes de l'interface

```
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);
  }
}
```

```
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);
  }
}
```

```
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();
  }
}
```

surcharge de méthode

```
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);
  }
}
```

```
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 {};
```

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

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

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

class NoSuchElementException extends Exception {};
```

pose et traitement externe

```
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();

      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);}
} } }
```

test et traitement local

```
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");
} }
```

```
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("Interru
      } }
      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");
} }
```

interface nécessaire

méthode obligatoire

création du thread

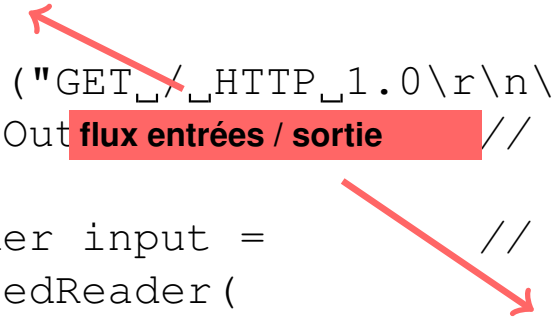
```
class T implements Runnable{
   public void run() {
      for (int i = 0; i<5; i++) {
         System.out.println("Processus␣léger␣T" + i);
         try { Thread.sleep(500);} // 500 msec = 1/2 s
         catch (InterruptedException e) {
            System.out.println("Interruption");
      } }
 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

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

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

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

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

      output.write("GET␣/␣HTTP␣1.0\r\n\r\n"); output.flush();
      cnx.shutdownOutput();          // fermeture partielle

      BufferedReader input =         // flux en lecture
        new BufferedReader(
        new InputStreamReader(cnx.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 cnx = null;
    try {
      cnx = new Socket("www.univ-paris13.fr",80);
      Writer output = new OutputStreamWriter(
      cnx.getOutputStream(), "8859_1");

      output.write("GET␣/␣HTTP␣1.0\r\n\r\n"); output.flush();
      cnx.shutdownOutput();          // fermeture partielle

      BufferedReader input =         // flux en lecture
        new BufferedReader(
        new InputStreamReader(cnx.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);}
}   }   }
```

connexion à un socket serveur

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

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

      output.write("GET␣/␣HTTP␣1.0\r\n\r\n"); output.flush();
      cnx.shutdownOut                          // fermeture partielle

      BufferedReader input =                   // flux en lecture
        new BufferedReader(
        new InputStreamReader(cnx.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.net.*; import java.io.*;

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

    public Main(int port) {
      try { serverSocket = new ServerSocket(port, 1);}
      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);
      while (true) {
         try { Socket socket = serverSocket.accept();
// code métier sur la connexion avec le client
         }
         catch (IOException e) {System.out.println(e);}
         finally {
             try { if (socket != null) socket.close();}
             catch (IOException e) {}
}  }  }  }
```

```
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);}
      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);
      while (true) {
         try { Socket socket = serverSocket.accept();
// code métier sur la connexion avec le client
         }
         catch (IOException e) {System.out.println(e);}
         finally {
             try { if (socket != null) socket.close();}
             catch (IOException e) {}
}  }  }  }
```

création d'un socket serveur

réception d'un client

- 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);
   }
}
```

`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 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);}
    }
}
```

Création du pool

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

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 );
  }
}
```

(slide suivant pour la classe `PolygoneRegulier`)

```
class PolygoneRegulier {
   int nbCotes, lgCotes;

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

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

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

Un `Stream` permet effectivement d'appliquer des fonctions sur un flux d'éléments.
Un flux peut être créé à partie 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 paralllél e :

```
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 Annotations en Java

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).

- Une annotation se pose avant n'importe quel modifieur (p.e. **`public`**) d'une classe, d'une méthode ... (en Java 8 : même un type !)
- Ele peut être combinée en une séquence d'annotations
- Une annotation peut 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("val")
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
$
```

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();
}}
```

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, ...).

Annotations prédéfinies ajoutées dans la version 1.6 :

`@Generated` : précise le code généré par un framework

```
public class MyClass{
    public void codePersonnel(){
    }
    @Generated(
        value = "ClasseGeneratrice",
        comments = "blablabla",
        date = "24 avril 2014"
    )
    public void codeGenere(){
    }
}
```

`@Resource` : spécifie comment la ressource est utilisable

```
@Resource(name = "MaListe",
    type = javax.jms.Queue,
    shareable = false,
    authenticationType = Resource.AuthenticationType.CONTAINER,
    description = "Une Liste"
)
private javax.jms.Queue maListe;
```

`@PostConstruct` et `@PreDestroy` : spécifie ce qu'il faut faire en création d'instance et lors de la destruction de l'instance

```
public class ClassePersonnelle{

    public ClassePersonnelle(){
    }

    @PostConstruct()
    public void loadDefaults(){
        // blabla
    }

    @PreDestroy()
    public void releaseResources(){
        // blabla
    }
}

}
```

`@DeclareRoles` : spécifie les autorisations

```
@DeclareRoles(value = {"Director", "Manager", "Others" })
public class LeaveService{
    @Resource
    private Context context;

    public void applyLeave(){
        // Any employee can apply for leave.
    }

    public void grantLeave(){   if(checkUserInRole()){
            // Grant Leave.
    }   }

    public void cancelLeave(){  if(checkUserInRole()){
            // Cancel Leave.
    }   }

    private boolean checkUserInRole(){
        if( (context.isCallerInRole("Director") )
            || (context.isCallerinRole("Manager")) ){
            return true;
        }
        return false;
}   }
```

Il existe un framework permettant d'utiliser des annotations de test : **The Checker Framework**

```
javac -processor
          org.checkerframework.checker.nullness.NullnessChecker
      MaClasse.java
```

Permet les tests suivants entre autres :

- `@NonNull` test si la variable est non nulle (bien pour les sockets !)
- `@MapKey` teste si la variable est une clé de map
- `@GuardedBy, @Holding` teste sur la synchronisation
- `@Regex` teste si une expression est régulière
- `@Format` teste le formatage d'une chaîne de caractères
- ...

```
public class UneClasse {
    ...
    void maMethode() {
        @NonNull Truc ref = toto;
    }
}
```

```
void helper1(@GuardedBy("MyClass.myLock") Object a) {
    a.toString(); // ILLEGAL: the lock is not held
    synchronized(MyClass.myLock) {
      a.toString();  // OK: the lock is held
}   }

@Holding("MyClass.myLock")
void helper2(@GuardedBy("MyClass.myLock") Object b) {
  b.toString(); // OK: the lock is held
}

void helper3(Object c) {
  helper1(c); // OK
  c.toString(); // OK: no lock constraints
}

void myMethod2(@GuardedBy("MyClass.myLock") Object e) {
  helper1(e);  // OK pass to a method without locking
  e.toString(); // ILLEGAL: the lock is not held
  synchronized (MyClass.myLock) {
    helper2(e);
    helper3(e);
} }
```

```
public @Regex String parenthesize(@Regex String regex) {
    return "(" + regex + ")";
}
```

```
@Format({FLOAT, INT})  String f;

    f = "%f␣%d";        // Ok
    f = "%s␣%d";        // OK, %s is weaker than %f
    f = "%f";           // warning: last argument is ignored
    f = "%f␣%d␣%s";     // error: too many arguments
    f = "%d␣%d";        // error: %d is not weaker than %f

String.format("%d", 42);                          // OK
String.format(Locale.GERMAN, "%d", 42);           // OK
String.format(new Object());                      // error (1a)
String.format("%y");                              // error (1a)
String.format("%2$s", "unused", "used");          // warning (1b)
```

Exemple : Traitement manuel à l'exécution (1)

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

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

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

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

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

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

Exemple : Traitement manuel à l'exécution (1)

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

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

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

l'annotation Audit sera conservée à l'exécution

```java
public class App
  @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.isAnnotationP                    urn;

   // Récupérer la donnée de l'ann                    er 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 p                                                    ramType;

   // Récupérer la mé
         Method metho                                                    mTypesArray);

   // 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 (Exceptio e) { audit("[Audit exception] sur l'appel de " + methodName);
      }
   }
   private static                                    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)

```
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 -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: {
}
```

Exemple : Traitement via le compilateur (2)

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

Exemple : Traitement via le compilateur (3)

```
import java.util.*;
import javax.annotati
import javax.lang.mod                    .lang.model.element.*;

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

   @Over                                 it(ProcessingEnvironment environment) {
      super.init(environment);
      this.environment = environment;
   }
...
```

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;
   }
...
```

classe-processeur par défaut

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

```
...
@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;
}
}
```

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

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;
   }
```

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.su                                    ;
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;
   }
```

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;
   }
```

méthode standard de visite d'un élément

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

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

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

3 Analyse XML

- XML, DTD, ...
- JAXB : XML/classe
- JAXP
  - SAX : Analyse de flux XML
  - DOM : vue Java d'un document XML

3 Analyse XML

- XML, DTD, ...
- JAXB : XML/classe
- JAXP
  - SAX : Analyse de flux XML
  - DOM : vue Java d'un document XML

XML : *eXtended Markup Language*

- format 'texte' structuré, portable, extensible
- Dérivé de SGML
- Standards associés :
  - XSL : eXtensible Stylesheet Language
  - XSLT : langage de transformation XSL
  - Xlink, Xquery : langages de requêtes sur documents XML
  - Xpath, Xpointer : langages d'accès aux parties

```
<?xml version="1.0" encoding="UTF8"?>
   <webapp
      version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/webapp_2_4.xsd">

      <displayname>hello1</displayname>
      <servlet>
         <displayname>index</displayname>
         <servletname>index</servletname>
         <jspfile>/index.jsp</jspfile>
      </servlet>
   </webapp>
```

Espace de noms en XML (*namespace*) :

- un espace de noms est un ensemble d'identifiants (éléments et attributs) à sémantique unique pour cet espace. On peut donc par exemple distinguer des attributs syntaxiquement identiques qui relèvent de 2 sémantiques distinctes.
- l'espace de noms est défini par un URI
- il est déclarable comme valeur de l'attribut `xmlns`
- `xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"` permet d'indiquer que tout attribut commençant par `xsi:` appartiendra à l'espace de noms

  `http://www.w3.org/2001/XMLSchemainstance`
- `xmlns="http://www.w3.org/2001/XMLSchemainstance"` permet d'indiquer que tout attribut sans préfixe appartiendra à l'espace de noms

  `http://www.w3.org/2001/XMLSchemainstance`

Différences DTD/XML Schema :

- DTD : pas de typage fort, 10 types de données prédéfinis

```
<!ELEMENT employee (shift+, homeaddress,hobbies*)>
<!ATTLIST shift id CDATA #REQUIRED>
```

- XML Schema :
  - 44 types de données prédéfinis
  - Même syntaxe que XML
  - Orienté objet
  - Gestion d'espaces de noms

```
<?xml version="1.0"?>
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.books.org" xmlns=http://www.books.org>

   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="Title" type="xsd:string"/>

   <xsd:element name="Author" type="xsd:string"/>
</xsd:schema>
```

JAXP : API Java pour l'analyse XML

- SAX (*Simple API for XML processing*) : analyse par flux
- DOM (*Document Object Model*) : construction d'un modèle représentant la donnée XML

- `javax.xml.parsers` pour les structures basiques
- `org.xml.sax` pour les algorithmes basiques sous SAX (et à utiliser en DOM)
- `org.w3c.dom` pour la structure de document XML

JAXB : 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, xbj*)
- la classe `JAXBContext` (de `javax.xml.bind`) intègre un analyseur/générateur XML qui fonctionne en utilisant les annotations des classes considérées

JAXB et traitement XML

Opérations disponibles :

- analyse (*unmarshall*) 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 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>
```

Exemple : Données de formation d'enseignement 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>
```

un élément XML

Exemple : Données de formation d'enseignement 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>
```

une formation :

a un intitulé (attribut obligatoire)

Exemple : Données de formation d'enseignement 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:sequence>
  </xsd:complexType>
  <xsd:element name="etudiant" type="typeEtudiant"/>
  <xsd:complexType name="type
    <xsd:attribute name="pren
    <xsd:attribute name="nom"
  </xsd:complexType>
  <xsd:element name="responsable" type="typeResponsable"/>
  <xsd:complexType name="typeResponsable">
   <xsd:attribute name="nom" type="xsd:string"/>
  </xsd:complexType>
 </xsd:schema>
```

une formation :

inclut un responsable

inclut des étudiants

Un exemple : M2PLS.xml

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

Génération de classe Java à partir de fichiers xsd :

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

Un exemple : M2PLS.xml

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

Génération de classe Java à partir de fichiers xsd :

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

pour ce schéma XML

dans ce paquetage (-p)

mis dans ce répertoire (-d)

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

Fichier généré :

```
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;
    }
...
```

```
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 leResp = uneFormation.getResponsable();
      System.out.println("Responsable : " + leResp.getNom());
...
```

```
import java.io.*;
import javax.xml.bind.*;
import up13.forma
public class Test
  public static v                    ng 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 : "
                                                    ntitule());
      TypeResponsable leR                           getResponsable();
      System.out.println(                           leResp.getNom());
...
```

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

```
...
// 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);}
  }
}
```

```
...
// création de données
      TypeFormation uneAutreFo                    ypeForm
      une                                  etIntitule("M2EID");
// Généra
      JAX                                  mation> uneAFJAXB =
                  (new upi3.formation.ObjectFactory())
                              .createFormation(uneAutreFormation);
      Marshaller marshaller = jaxbContext.createMarshaller();
      marshaller.marshal( uneAFJAXB, System.out );
    } catch (JAXBException e) {System.err.println(e);}
  }
}
```

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

```
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(
                     "Définition␣de␣formation␣invalide␣:␣"
                              + locator.getURL());
            System.out.println("Erreur␣:␣" + ve.getMessage());
            System.out.println("Colonne␣" +
                              locator.getColumnNumber() +
                              ",␣ligne␣"
                              + locator.getLineNumber());
         }
         return true;
       }
   }
```

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

public class FormationValidationEventHandler
                        implements ValidationEventHandler{

      public boo                                                {
        if (ve.getSeverity()  ValidationEvent.FATAL_ERROR ||
            ve .getSeverity()==ValidationEvent.ERROR){
            ValidationEventLocator  locator = ve.getLocator();
            //
            System.out.println(
                     "Définition de formation invalide : "
                              + locator.getURL());
            System.out.println("Erreur : " + ve.getMessage());
            System.out.println("Colonne " +
                              locator.getColumnNumber() +
                              ", ligne "
                              + locator.getLineNumber());
        }
        return true;
      }
   }
```

Gestionnaire d'événements pour l'analyseur XML (cas d'erreurs, ...)

3 Analyse XML

- XML, DTD, ...
- JAXB : XML/classe
- JAXP
  - SAX : Analyse de flux XML
  - DOM : vue Java d'un document XML

JAXP :

- `javax.xml.parsers` inclut les classes :
  - `SAXParser` et sa fabrique `SAXParserFactory`
  - `DocumentBuilder` et sa fabrique `DocumentBuilderFactory`
- `org.xml.sax` inclut principalement les interfaces de gestion de données XML :
  - `XMLReader` pour les lecteurs
  - `DTDHandler` pour la conformité aux DTD
  - ...

SAX le principe :

- un objet *reader* effectue la lecture d'un flux de données (fichier, socket, ...) en constituant à partir de codes liés à l'application des objets représentant le contenu XML
- au début/à la fin du document, à chaque début/fin de noeuds (*element*) de l'arbre XML lu, l'objet-lecteur fait appel à une méthode spécifique d'un gestionnaire de contenu.
- la mise en oeuvre consiste au minimum à :
  - définir un gestionnaire de contenu, i.e. les méthodes décrivant les opérations à effectuer pour la construction des objets selon les données XML (début/fin d'élément, ...)
  - définir un gestionnaire d'erreurs (pour traiter correctement les erreurs du flux XML par rapport à ce qui est attendu)
  - définir un objet `SAXParser` encapsulant les objets précédents et terminant la configuration de l'analyseur-lecteur.

Mise en oeuvre concrète :

- création de l'objet fabrique d'analyseurs-lecteurs comme instance de `SAXParserFactory`
- configuration de cet objet : gestion d'espace de noms, validation par une DTD ou un schéma XML, ...
- création d'un analyseur-lecteur à partir de la fabrique précédente comme instance de `SAXParser`
- récupération, à partir de l'objet précédent, d'un lecteur `XMLReader`
- création d'un gestionnaire de contenu, d'un gestionnaire d'erreurs, d'un gestionnaire de schéma spécifiques à l'application
- configuration de ce lecteur : gestionnaire de contenu, gestionnaire d'erreurs, gestionnaire de schémas
- lancement de l'analyse du flux XML

Exemple (1) : EBM.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE EBM [
   <!-- DTD basique pour la sauvegarde d'EBM au format XML -->

   <!ELEMENT EBM (GraphicalData,ComposantPlus,ComposantsMoins)>
   <!ELEMENT GraphicalData (GraphicalSizeData,PointOrigine)>
   <!ELEMENT GraphicalSizeData EMPTY>
   <!ATTLIST GraphicalSizeData
      largeurMin CDATA #REQUIRED
      largeur CDATA #REQUIRED
      hauteur CDATA #REQUIRED
   >
   <!ELEMENT PointOrigine EMPTY>
      <!ATTLIST PointOrigine
         x CDATA #REQUIRED
         y CDATA #REQUIRED
      >
   <!ELEMENT ComposantPlus (GraphicalData,Ressources)>
   <!ELEMENT ComposantsMoins (ComposantMoins*)>
   <!ELEMENT ComposantMoins (GraphicalData,Ressources)>
   <!ELEMENT Ressources (Ressource*)>
   <!ELEMENT Ressource (GraphicalData,Etiquette)>
   <!ELEMENT Etiquette (GraphicalData)>
   <!ATTLIST Etiquette Valeur CDATA #REQUIRED>
]>
```

Exemple (2) : un flux de donnée XML

```
<EBM>
   <GraphicalData>
      <GraphicalSizeData largeurMin="30" largeur="40" hauteur="25">
         <PointOrigine x="83" y="123"/>
      </GraphicalData>
   <ComposantPlus>
      <GraphicalData>
         <GraphicalSizeData largeurMin="30" largeur="40" hauteur="25"/>
         <PointOrigine x="83" y="123"/>
      </GraphicalData>
      <Ressources>
         <Ressource>
            <GraphicalData>
               <GraphicalSizeData largeurMin="30" largeur="40" hauteur="25"/>
               <PointOrigine x="83" y="123"/>
            </GraphicalData>
            <Etiquette Valeur="a">
            ...
            </Etiquette>
         </Ressource>
      </Ressources>
   </ComposantPlus>
   <ComposantsMoins>
      <ComposantMoins>
      ...
      </ComposantMoins>
   </ComposantsMoins>
</EBM>
```

Classe principale initialisant les objets et lançant l'analyse

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import java.io.*;

public class SAXEBMReader {
...
  public SAXEBMReader (String fileName, BM_graph bm_graph)
         throws IOException {
    ...
   // Création de la fabrique d'analyseurs-lecteurs
     SAXParserFactory spf = SAXParserFactory.newInstance();
     spf.setNamespaceAware(true); spf.setValidating(false);

     try {
   // création d'un analyseur
       SAXParser saxParser = spf.newSAXParser();
   // on récupère le lecteur
       XMLReader xmlReader = saxParser.getXMLReader();
   // configuration du lecteur
       xmlReader.setContentHandler
                 (new SAXEBMReaderProcess(getBM_graph()));
       xmlReader.setErrorHandler
                 (new MyErrorHandler(System.err));
...
}
```

```
...
   // analyse du flux de données XML
         xmlReader.parse(fileNale.toURL().toString());
      }
      catch (SAXException saxe) {
          System.out.println("SAX : " + saxe);
      }
      catch (ParserConfigurationException pce) {
          System.out.println("conf : " + pce);
      }
   }
...
}
```

Gestionnaire d'erreurs

```
private static class MyErrorHandler implements ErrorHandler {
   private PrintStream out;
   MyErrorHandler(PrintStream out) {this.out = System.out;}

   private String getParseExceptionInfo(SAXParseException spe) {
      String systemId = spe.getSystemId();
      if (systemId == null) {systemId = "null";}
      String info = "URI=" + systemId + "␣Line="
                  + spe.getLineNumber() + ":␣" + spe.getMessage();
      return info;
   }

   public void warning(SAXParseException spe) throws SAXException {
      out.println("Warning:␣" + getParseExceptionInfo(spe));
   }

   public void error(SAXParseException spe) throws SAXException {
      String message = "Error:␣" + getParseExceptionInfo(spe);
      throw new SAXException(message);
   }

   public void fatalError(SAXParseException spe) throws SAXException {
      String message = "Fatal␣Error:␣" + getParseExceptionInfo(spe);
      throw new SAXException(message);
   }
}
```

Gestionnaire de contenu XML

```
import org.xml.sax.*; import org.xml.sax.helpers.*;
import java.util.*;

public class SAXEBMReaderProcess extends DefaultHandler {
   ...

   public SAXEBMReaderProcess (BM_graph bm_graph) {
      super();
      setBM_graph(bm_graph);
   }

   // Opérations en début de document
   public void startDocument() throws SAXException {
   }

   // Opérations en fin de lecture de document
   public void endDocument() throws SAXException {
   }

...
```

```
...
   // Opérations en début de lecture d'élément
   public void startElement(String namespaceURI, String localName,
                            String qName, Attributes atts)
               throws SAXException {
      String key = localName;
   // opérations selon l'attribut lu
      if (localName.equals("BM")) { ... };
      if (localName.equals("GraphicalSizeData")) { ... };
      ...
   }

   // Opérations en fin de lecture d'élément
   public void endElement(String namespaceURI, String localName,
                          String qName) throws SAXException {
      String key = localName;
   // opérations selon l'attribut lu
      if (localName.equals("EBM")) { ... };
      if (localName.equals("Ressource_Extremite")) { ... };
      ...
   }
...
}
```

Sur les arguments des méthodes `startElement` et `endElement` :

- `namespaceURI` : l'URI de l'espace de noms
- `localName` : nom sans l'URI préfixe
- `qName` : nom avec l'URI préfixe
- `attributes` : tableau des attributs présents dans l'élément

DOM le principe :

- un objet *reader* effectue la lecture d'un flux de données (fichier, socket, ...) en constituant des objets en miroir du contenu XML
- la mise en oeuvre est assez similaire au cas SAX sans gestionnaire de contenu.
- une fois les objets construits, ils sont accessibles "directement".

Mise en oeuvre concrète :

- création de l'objet fabrique d'analyseurs-lecteurs comme instance de `DocumentBuilderFactory`
- configuration de cet objet : gestion d'espace de noms, validation par une DTD ou un schéma XML, ...
- création d'un analyseur-lecteur à partir de la fabrique précédente comme instance de `DocumentBuilder`
- Pas de reader à extraire
- création d'un gestionnaire d'erreurs, d'un gestionnaire de schéma spécifiques à l'application
- configuration du lecteur : gestionnaire d'erreurs, gestionnaire de schémas
- lancement de l'analyse du flux XML qui renvoie une instance de `Document`
- consultation des noeuds du document.

- `Document` et `Node` sont deux interfaces (instanciées dans le paquetage w3c)
- des attributs de `Node` permettent d'accéder aux éléments et attributs.

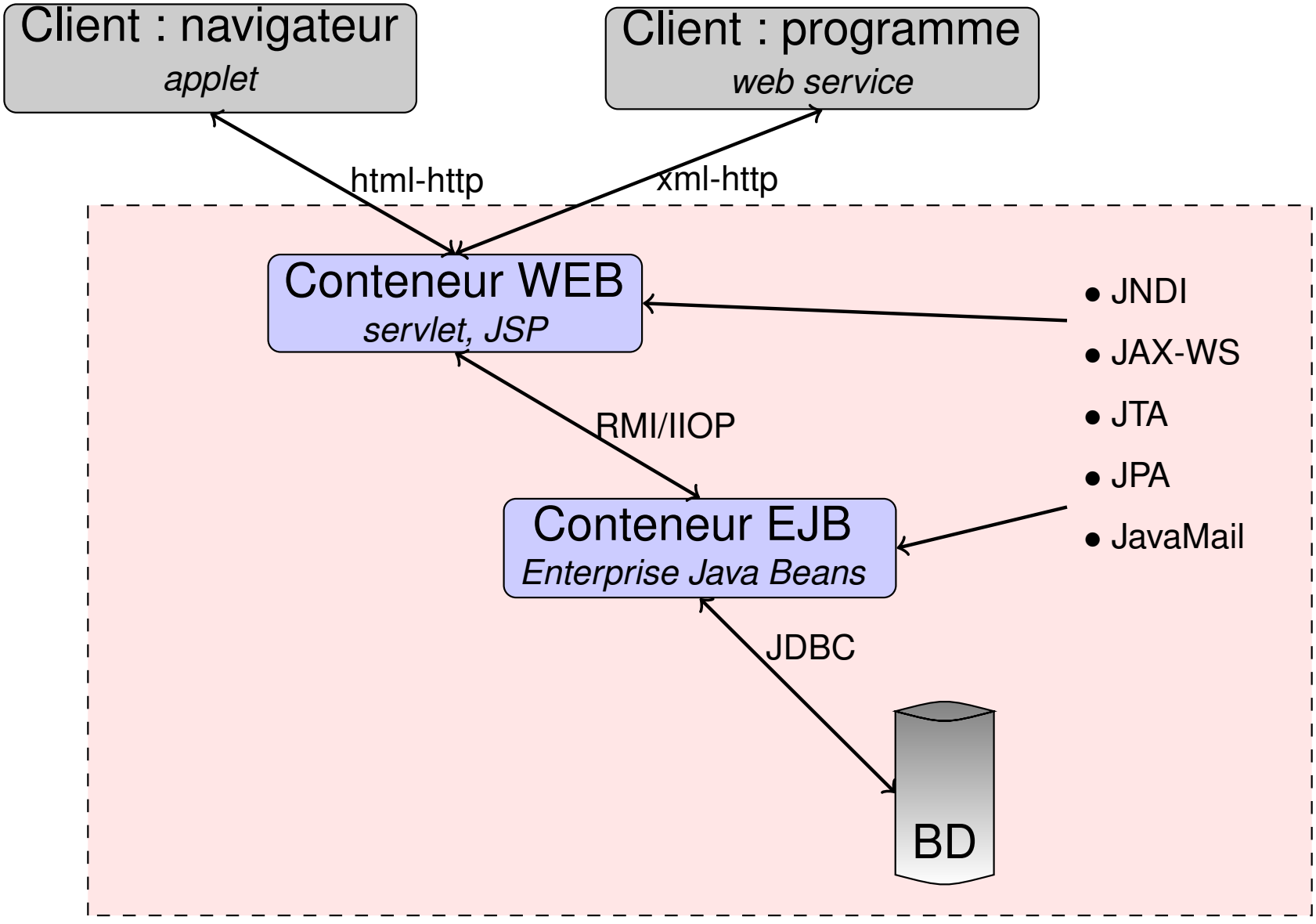
Analyseur DOM

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;
import java.io.*;
...
public static void main(String[] args) throws Exception {
   String filename = args[0];

// Création de la fabrique
   DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
   dbf.setNamespaceAware(true);
   dbf.setValidating(false);
   dbf.setIgnoringComments(false);
   dbf.setIgnoringElementContentWhitespace(false);

// Création du lecteur DOM
   DocumentBuilder db = dbf.newDocumentBuilder();
// gestionnaire d'erreurs
   db.setErrorHandler(new MyErrorHandler());

// analyse du document XML
   Document doc = db.parse(new File(filename));

... echo(doc);
}
```

```
...
private void echo(Node n) {
    int type = n.getNodeType();
    switch (type) {
        case Node.ATTRIBUTE_NODE:
            out.print("ATTR:"); printlnCommon(n);
        break;
        case Node.CDATA_SECTION_NODE:
            out.print("CDATA:"); printlnCommon(n);
        break;
        case Node.COMMENT_NODE:
            out.print("COMM:"); printlnCommon(n);
        break;
        case Node.ELEMENT_NODE:
            out.print("ELEM:"); printlnCommon(n);
            NamedNodeMap atts = n.getAttributes();
            indent += 2;
            for (int i = 0; i < atts.getLength(); i++) {
                Node att = atts.item(i); echo(att);
            }
            indent -= 2;
        break;
    }
}
...
```

JEE / **J2EE** (*Java Enterprise Bean*) : Environnement permettant le développement et le déploiement d'applications réparties multi-plateformes (mais en Java)
Objectif : intégrer dans un unique environnement

- une machine virtuelle permettant l'exécution de programmes
- une facilité de distribution du code sur un parc de machines, avec possibilité de répartition de charges
- un interfaçage transparent avec des bases de données
- un interfaçage avec des éléments externes (sites web, services web)


Client : navigateur
applet
Client : programme
web service
html-http
xml-http
Conteneur WEB
servlet, JSP
• JNDI
• JAX-WS
• JTA
• JPA
• JavaMail
RMI/IIOP
Conteneur EJB
Enterprise Java Beans
JDBC
BD

Chaque conteneur

- gère un ensemble de fichiers (.java et .class)
- utilise des *descripteurs* de compilation et de déploiement (format XML)
- remplit un ensemble de fonctionnalités plus ou moins transparentes pour l'utilisateur :
  - nommage et annuaire
  - notification et envoi de messages
  - cycle de vie
  - stockage et archivage
  - système transactionnel
  - contrôle de sécurité
  - administration d'objets (pool, équilibrage de charges)

Technologies (1) :

- Composants web :
  - Servlets : code Java de génération de pages HTML ou XML
  - JSP : code à compiler à la demande en servlet
- Composants EJB :
  - Beans de session : objets liés à une (session de) requêtes de clients (p.e. panier)
  - Beans entité : objets modélisant un objet de BD
  - Beans message : objets gérant les communications asynchrones

Technologies (2) :

- Services :
  - JNDI : *Java Naming and Directory Interface* (lien vers un LDAP, ...)
  - JAX-WS : *Java API for XML Web Services*
  - JTA : *Java Transaction API* (gestion des transactions distribuées)
  - JPA : *Java Persistence API* (gestion de la persistence de contenu de session hors BD)
  - JDBC : *Java Database Connectivity*
- Communication :
  - HTTP, TCP/IP, SSL (HTTPS)
  - RMI/IIOP : applications distribuées
  - JMS et JavaMail : communication asynchrone (messagerie)

Environnements :

* Environnements de développement (IDE) : Netbeans, Eclipse, JBuilder, ...

* Environnements complets de déploiement : GlassFish (ex-JWSDP), .NET, WebObjects, BEA, Oracle Application Server, IBM Websphere, ...

* Environnements partiels combinables (en partie) :

- conteneur web =
  - écouteur http : Apache httpd, Grizzly
  - gestionnaire jsp/servlet : (Jakarta Apache) Tomcat, Winstone
  - gestionnaire web services : Metro, BEA WebLogic, Server HTTP IIS, ...
- conteneur EJB = dépend de l'environnement complet

Java Servlet : base des cadres WebServlet, JSF, JSP

- Servlet =
  - code Java
  - init / **service** / destroy
- JSF (*Java Server Faces*) =
  - code XHTML
  - transformé en code servlet
  - Framework type MVC incluant une gestion clients avec états
  - Contrôle par `FacesServlet`
  - Basé composants
  - affichage via Facelets (ou JSP, ou ...)
- JSP (*Java Server Pages*) =
  - code XHTML
  - transformé en code servlet
  - les directives, scripts, actions sont de la forme
    `<%type contenu%>` ou
    `<jsp:type contenu> ... </jsp:type>`
- dans les 3 cas (et surtout servlet), vérification de la syntaxe en sortie impossible !

Servlet :

- Une servlet est exécutée côté serveur
- elle doit être capable de gérer les requêtes provenant de clients web
- la bibliothèque de classes est dépendant du conteneur web (donc pas en standard dans JDK mais dans tomcat, glassfish, ...) : `javax.servlet.*`

L'interface `Servlet` fixe la spécification, i.e. les méthodes à implanter :

- **`void`** `init(ServletConfig sc)` doit être appelé lors d'une demande du conteneur (après création d'une instance). La configuration est principalement le *contexte*, i.e. la liste des paramètres et leur valeur.
- **`void`** `service(ServletRequest req, ServletResponse res)` méthode principale devant gérer les requêtes clientes.
- **`void`** `destroy()` doit être appelé par le conteneur lorsque les appels `service()` sont terminés ou après un laps de temps.

Requête / Réponse :

- `ServletRequest` : interface spécifiant l'accès au flux d'entrée (`getInputStream()`)et aux paramètres d'une requête (`getAttribute(String name)`)
- `ServletResponse` : interface spécifiant le flux de sortie (`setEncoding()`, `getOutputStream()`, ...)
- on peut spécifier une classe par protocole (p.e. `HttpServlet` pour http), dépend du conteneur web.

Cas de HTTP :

- `javax.servlet.http.HttpServlet` définit un ensemble de méthodes dédiées au protocole http.
- la méthode `service()` est définie pour analyser le flux d'entrée (qui doit être une requête HTTP : get, post, ...), et en fonction de la requête concrète http appeler l'une des méthodes `doGet()`, `doPost()`, ...
- les méthodes `doXXX()` peuvent être redéfinies dans une sous-classe de `HttpServlet` selon les besoins de l'application.

Exemple :

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

import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldExample extends HttpServlet {
   public void doGet(HttpServletRequest request,
                                 HttpServletResponse response)
      throws IOException, ServletException
   {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println("<html>"); out.println("<head>");
   String title = "M2␣Informatique";
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body␣bgcolor=\"white\">");
   out.println("<a␣href=\"http:\/\/www.univparis13.fr\">");
   out.println("Universit&eacute;␣Paris␣13</a>");
   out.println("<h1>" + title + "</h1>");
   out.println("</body>");
   out.println("</html>");
   }
}
```

Exemple :

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

import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldExample extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
      throws IOException, ServletException
   {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println("<html>"); out.println("<head>");
   String title = "M2 Informatique";
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body bgcolor=\"white\">");
   out.println("<a href=\"http:\/\/www.univparis13.fr\">");
   out.println("Universit&eacute; Paris 13</a>");
   out.println("<h1>" + title + "</h1>");
   out.println("</body>");
   out.println("</html>");
   }
}
```

superclasse pour les servlets conformes au protocole Http

code pour une requête **GET**

Exemple :

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

import javax.servlet.*;
import javax.servlet.http.*
public class HelloWorldExample extends HttpServlet {
   public void doGet(HttpServletRequest request,
                                    HttpServletResponse response)
      throws IOException, ServletException
   {
   respons                                ext/ht
   PrintWriter out = response.getWriter();

   out.println("<html>"); out.println("<head>");
   String title = "M2 Informatique";
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body bgcolor=\"white\">");
   out.println("<a href=\"http:\/\/www.univparis13.fr\">");
   out.println("Universit&eacute; Paris 13</a>");
   out.println("<h1>" + title + "</h1>");
   out.println("</body>");
   out.println("</html>");
   }
}
```

contient les données envoyées par le client

écriture du début de page html

contient la spécif pour l'envoi (dont le flux de sortie)

Mise en place dans un environnement J2EE :

- Tous les fichiers dans `MonAppli.war`
- `MonAppli.war/Web Pages/` : contient les fichiers de configuration et les fichiers HTML (standard) ou XHTML (JSF)
- `MonAppli.war/Web Pages/WEB-INF/web.xml` : déclaration des mappings, initialisation, propriétés pour le serveur web
- `MonAppli.war/src/**` : contient les codes de servlets et de filtres

L'URL d'une requête http est de la forme suivante :

`http://[host]:[port][request-path]?[query-string]`

Le 'request path' est composé de :

- Context path : chemin de base
- Servlet path : nom de la servlet
- Path info : autres informations

Par exemple :

`http://localhost:37002/MonAppli/MaServlet`

Récupération de ces parties avec les méthodes `getContextPath`, `getServletPath`, `getPathInfo` sur `HttpServletRequest`.

Le ‘context path’ est positionné dans le fichier `glassfish-web.xml` si GlassFish est utilisé.

Il est aussi possible de le spécifier dans l’onglet `Properties` du menu associé à `MonAppli-war` dans Netbeans.

Déclaration d'une servlet :

- Soit la classe servlet + fichier `web.xml` :

```
<web-app version="3.1"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ... >
    <servlet>
        <servlet-name>HWE</servlet-name>
        <servlet-class>pkg.HelloWorldExample</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HWE</servlet-name>
        <url-pattern>/Bonjour</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout> 30 </session-timeout>
    </session-config>
</web-app>
```

- Soit avec une annotation :

```
@WebServlet(name = "HWE", urlPatterns = {"/Bonjour"})
public class HelloWorldExample extends HttpServlet {
...
}
```

- Un **filtre** est une classe Java qui s'interpose entre le client (en fait le conteneur web) et la servlet gérant la requête.
- Un filtre est applicable à un pattern de servlets (expression régulière sur les noms de servlets) : spécifié comme annotation ou dans le fichier `web.xml`
- Sur une servlet, peut être appliquée une **chaîne** de filtres.
- Un filtre sert en entrée et en sortie !
- Un filtre peut modifier et/ou compléter une requête ou une réponse (ou conserver de l'info, ou ...)
- Un filtre peut servir à
  - vérifier l'authentification du requêteur
  - faire du log
  - convertir des formats de données
  - ...

un filtre doit implanter l'interface `Filter`, donc en particulier instancier les méthodes :

- **`void`** `destroy()` : destruction de l'instance de filtre.
- **`void`** `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)` : appelé par le conteneur web si le filtre porte sur le pattern de requête
- **`void`** `init(FilterConfig filterConfig)` : initialisation du filtre

Filtre basique :

```
import ...

@WebFilter(filterName = "MonFiltre", urlPatterns = {"/Machin*"},
           initParams = {@WebInitParam
                  (name = "unAttribut", value = "saValeur")}))
public class MonFiltre implements Filter {
    private FilterConfig filterConfig = null;

    public MonFiltre() {}

    public void destroy() {}

    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }

    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
            throws IOException, ServletException {
        // le code du filtre
    }
}
```

`doFilter()` simple :

```
...
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
        throws IOException, ServletException {

    doBeforeProcessing(request, response);
    chain.doFilter(request, response);
    doAfterProcessing(request, response);
}

private void doBeforeProcessing(ServletRequest request,
                                ServletResponse response)
        throws IOException, ServletException {
    // blabla, p.e. log de la requpete avant traitement
}

private void doAfterProcessing(ServletRequest request,
                               ServletResponse response)
        throws IOException, ServletException {
    // blabla, p.e. log de la requete après traitement

     PrintWriter respOut = new PrintWriter(response.getWriter());
     respOut.println("<p><b>Ajout␣par␣mon␣filtre</b>");
}
...
```

`doFilter()` avec gestion d'exceptions :

```
...
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
        throws IOException, ServletException {

    doBeforeProcessing(request, response);
    Throwable problem = null;
    try {
        chain.doFilter(request, response);
    } catch (Throwable t) {
        problem = t;
        t.printStackTrace();
    }
    doAfterProcessing(request, response);

    if (problem != null) {
        if (problem instanceof ServletException) {
            throw (ServletException) problem;
        }
        if (problem instanceof IOException) {
            throw (IOException) problem;
        }
        // envoi d'un texte html mentionnant l'erreur au client
    }
}
```

`doFilter()` avec modification du flux client :

```
...
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
        throws IOException, ServletException {

  HttpServletRequest httpRequest=(HttpServletRequest)request;
  if (httpRequest.getContentType() == null) {
    newHttpRequest=new HttpServletRequestWrapper(httpRequest){
      @Override public String getContentType(){
        return "application/truc";
      }
    };
  }
  chain.doFilter(newHttpRequest,response);
}...
```

Enterprise Java Beans (EJB) (à ne pas confondre avec les *Beans* utilisés en Java Graphique) :

- un bean est un objet d'une machine virtuelle encapsulant une partie de la logique métier et interagissant avec d'autres beans ou "le monde extérieur" (clients, BD, ...)
- l'interaction (i.e. l'appel) entre beans est normalisée, masquée pour le programmeur car faisant partie des plate-formes de déploiement
- l'implantation des interactions inter-beans doit permettre la gestion par pool d'esclaves (multi-threads) et distribution de charge (multi-VM, RMI) : chaque type-classe de bean est géré comme un pool de threads.

- 3 types de beans :
  - Beans **session** : interagissent avec des clients, pas de persistence de données entre session (voire entre appels)
  - Beans **entité** : objets avec persistence de données, lien avec BD
  - Beans **orientés message** : objets gérant les communications asynchrones

Le principe des beans entité est maintenant spécifié comme la *Java Persistence API*.

Exemples de schémas d'activation :

- *client HTML :*
  - la requête arrive via le serveur http à une servlet
  - la servlet requiert un traitement par un bean session
  - si nécessaire, le cookie permet d'identifier les données requises pour initialiser le bean session
  - le résultat du code métier (dans le bean session) est utilisé pour construire par la servlet la réponse
  - le code de la servlet ne contient que la mise en forme des données
- *client programme (C ou autre) :*
  - via Corba, un appel à un bean session
  - le bean session active un ou plusieurs beans entité
  - chaque bean entité est une vue de base de données
  - la maintenance de la BD est gérée par les beans entité
  - le calcul métier est effectué par le bean session

Eléments sur la structure sous-jacente (1) : (il s'agit d'une possibilité d'implantation dans un framework)

1. Chaque classe de bean est gérée par un pool de threads
2. Une activation (session, ...) est une suite de demandes de calcul sur un contexte particulier
3. la "durée de vie" d'un contexte est longue au regard des requêtes et appels

- donc il est utile de sauvegarder temporairement chaque contexte (avec un identifiant type cookie), i.e. avoir une BD dédiée et des objets-coquilles
- du coup, il faut des objets gérant les objets contenant le code métier, les (ré)initialisant, les endormant en sauvegardant le contexte, ...
- Donc : pour chaque classe d'**objets**(-coquille), une classe d'objets d'**accueil** (dont le code est générique)

Eléments sur la structure sous-jacente (2) :

1. l'ensemble doit être multi-VM, donc accessible sous RMI via JNDI (i.e. nom publicisé), de même pour l'objet-utilisateur

   - Donc il faut **4 interfaces** : pour un objet et son objet d'accueil, en séparant ce qui est utilisable localement de ce qui est utilisable à distance. Les contenus des interfaces sont a priori génériques.

1. le déploiement doit être spécifié de telle sorte que ce soit utilisable à chaud.

   - Donc **1** fichier **descripteur de déploiement** (en XML) (en sus d'un fichier descripteur de compilation si nécessaire)

Exemple (1) :

```
public interface HelloRemote extends javax.ejb.EJBObject
   { public String hello() throws java.rmi.RemoteException; }
```

```
public interface HelloLocal extends javax.ejb.EJBLocalObject
   { public String hello(); }
```

```
public interface HelloRemoteHome extends javax.ejb.EJBHome
   { Hello create()
     throws java.rmi.RemoteException, javax.ejb.CreateException; }
```

```
public interface HelloLocalHome extends javax.ejb.EJBLocalHome
   { HelloLocal create() throws javax.ejb.CreateException; }
```

```
public class HelloBean implements javax.ejb.SessionBean {
   private SessionContext ctx;
   public void ejbCreate() {...}
   public void ejbRemove() {...}
   public void ejbActivate() {...}
   public void ejbPassivate() {...}
   public void setSessionContext(javax.ejb.SessionContext)
         {this.ctx=ctx;}
// contenu concret "métier"
   public String hello() { return "Hello";}
}
```

Exemple (1) :

```
public interface HelloRemote extends javax.ejb.EJBObject
   { public String hello() throws java.rmi.RemoteException; }
```

Interface distante : utilisable par un client à travers RMI.

```
public interface HelloLocal extends javax.ejb.EJBLocalObject
   { public String hello(); }
```

Interface locale : utilisable par un client de la VM.

```
public interface HelloRemoteHome extends javax.ejb.EJBHome
   { Hello create()
     throws java.rmi.RemoteException, javax.ejb.CreateException; }
```

```
public interface HelloLocalHome extends javax.ejb.EJBLocalHome
   { HelloLocal create() throws javax.ejb.CreateException; }
```

```
public class HelloBean implements javax.ejb.SessionBean {
   private SessionContext ctx;
   public void ejbCreate() {...}
   public void ejbRemove() {...}
   public void ejbActivate() {...}
   public void ejbPassivate() {...}
   public void setSessionContext(javax.ejb.SessionContext)
         {this.ctx=ctx;}
// contenu concret "métier"
   public String hello() { return "Hello";}
}
```

Exemple (1) :

```
public interface HelloRemote extends javax.ejb.EJBObject
   { public String hello() throws java.rmi.RemoteException; }
```

```
public interface HelloLocal extends javax.ejb.EJBLocalObject
   { public String hello(); }
```

```
public interface HelloRemoteHome extends javax.ejb.EJBHome
   { Hello create()
     throws java.rmi.RemoteException, javax.ejb.CreateException; }
```

Interface d'accueil distante : "création" d'objets-coquilles à travers RMI.

```
public interface HelloLocalHome extends javax.ejb.EJBLocalHome
   { HelloLocal create() throws javax.ejb.CreateException; }
```

```
public class HelloBean implements javax.ejb.SessionBean {
   private SessionContext ctx;
   public void ejbCreate() {...}
   public void ejbRemove() {...}
   public void ejbActivate() {...}
   public void ejbPassivate() {...}
   public void setSessionContext(javax.ejb.SessionContext)
         {this.ctx=ctx;}
// contenu concret "métier"
   public String hello() { return "Hello";}
}
```

Interface d'accueil locale : "création" d'objets-coquilles sur la VM.

Exemple (1) :

```
public interface HelloRemote extends javax.ejb.EJBObject
   { public String hello() throws java.rmi.RemoteException; }
```

```
public interface HelloLocal extends javax.ejb.EJBLocalObject
   { public String hello(); }
```

```
public interface HelloRemoteHome extends javax.ejb.EJBHome
   { Hello create()
     throws java.rmi.RemoteException, javax.ejb.CreateException; }
```

```
public interface HelloLocalHome extends javax.ejb.EJBLocalHome
   { HelloLocal create() throws javax.ejb.CreateException; }
```

```
public class HelloBean implements javax.ejb.SessionBean {
   private SessionContext ctx;
   public void ejbCreate() {...}
   public void ejbRemove() {...}
   public void ejbActivate() {...}
   public void ejbPassivate() {...}
   public void setSessionContext(javax.ejb.SessionContext)
         {this.ctx=ctx;}
// contenu concret "métier"
   public String hello() { return "Hello";}
}
```

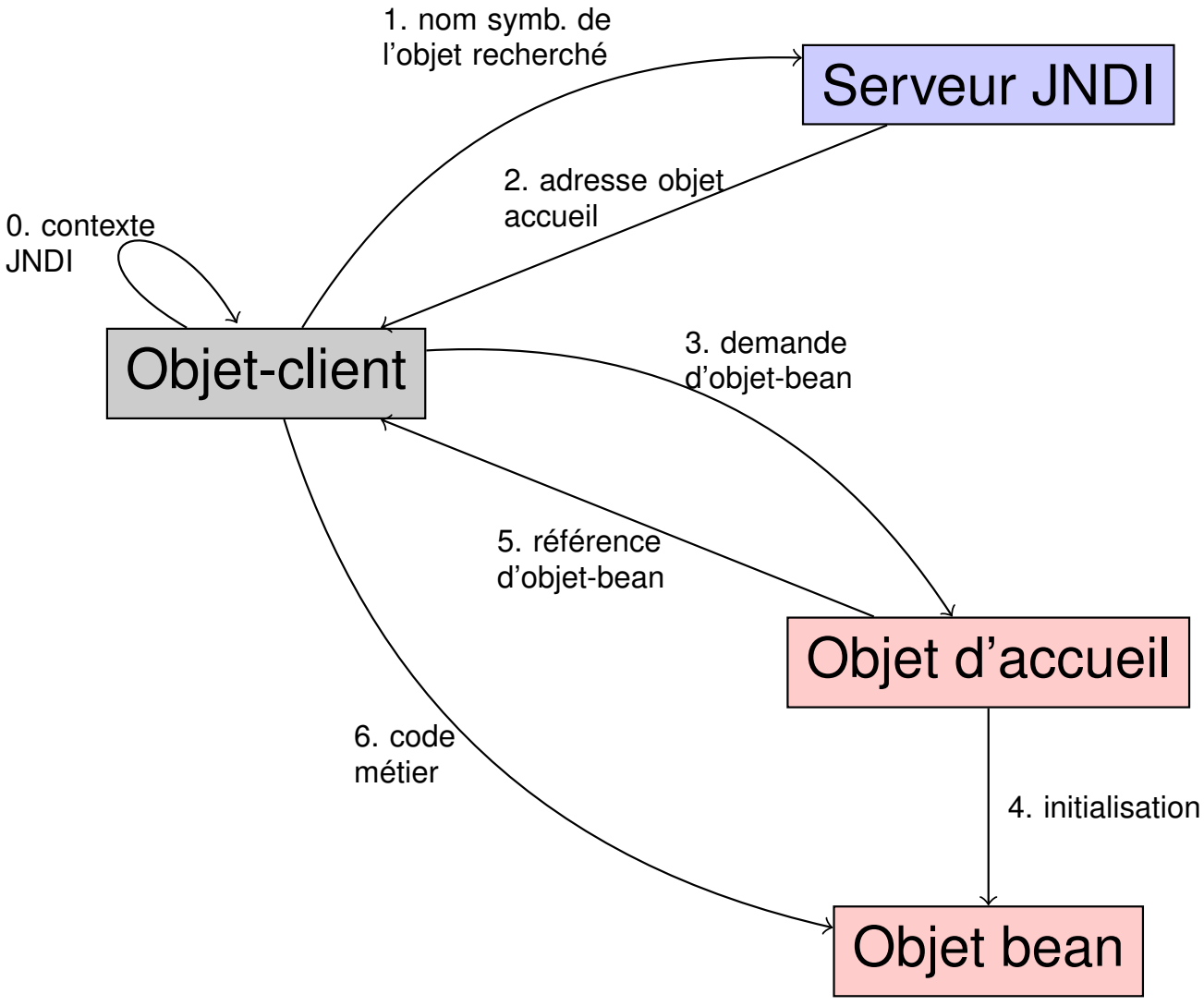
Code du bean de type session

Exemple (2) :

```
<!-- ejb-jar.xml (descripteur de deploiement) -->
<!DOCTYPE ejb-jar ... ejb-jar.dtd>
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>Hello</ejb-name>
      <home>HelloRemoteHome</home>
      <remote>HelloRemote</remote>
      <local-home>HelloLocalHome</local-home>
      <local>HelloLocal</local>
      <ejb-class>HelloBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Exemple (3) :

```
// HelloClient.java (exemple de code client)
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.Properties;

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

// informations pour usage de JNDI
    Context ctx = new InitialContext(System.getProperties());

// récupération de l'objet d'accueil
    Object obj = ctx.lookup("HelloHome");
// transtypage spécifique
    HelloHome helloHome = (HelloHome)
            PortableRemoteObject.narrow (obj, HelloHome.class);

// objet EJB servant d'interface avec le (vrai) bean
    Hello hello = helloHome.create();

// appel des méthodes "métier"
    System.out.println(hello.hello());

// fin d'usage: libération de l'objet EJB
    hello.remove();
  }
}
```


1. nom symb. de
l'objet recherché
Serveur JNDI
2. adresse objet
accueil
0. contexte
JNDI
Objet-client
3. demande
d'objet-bean
5. référence
d'objet-bean
Objet d'accueil
6. code
métier
4. initialisation
Objet bean

Beans session :

- durée de vie : celle de l'appel
- 3 sous-types :
  - `@Stateless` : aucune donnée de session. Exemple : envoi de l'heure
  - `@Stateful` : il existe des données de session, mécanisme de type cookie pour réinitialiser le contexte. Exemple : chariot d'achat
  - `@Singleton` : pour les accès concurrents sur une application
- 3 cas d'appel d'un "client" :
  - `@Local` : interne à la VM
  - `@Remote` : accessible via JNDI par des "clients" extérieurs
  - `@WebService` : accessible comme web service

Exemple de bean session *stateless* avec tags de précompilation :

```java
// interface pour RMI
import javax.ejb.Remote;
@Remote
public interface Hello {
   String hello();
}
```

```java
// interface pour local
import javax.ejb.Local;
@Local
public interface Hello {
   String hello();
}
```

```java
// code du bean session
import javax.ejb.Stateless;
@Stateless // (mapped-name="nomPublic")
public class HelloBean implements Hello {
   public String hello() { return ("Hello world !"); }     }
```

```java
// code d'un objet-client de ce bean session
import java.io.*;
import javax.ejb.EJB;
import javax.servlet.*; import javax.servlet.http.*;

public class TestServlet extends HttpServlet {
   @EJB // (mapped-name="nomPublic")
   private Hello hello;
   public void service (HttpServletRequest request,
                        HttpServletResponse response)
         throws ServletException, IOException {
      response.getWriter().println(hello.hello());
}   }
```

Exemple sans interface :

```
// code du bean session
import javax.ejb.Stateless;
@Stateless // (mapped-name="nomPublic")
@LocalBean
public class HelloBean {
   public String hello() { return ("Hello world !"); }
}
```

```
// code d'un objet-client de ce bean session
import java.io.*;
import javax.ejb.EJB;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestServlet extends HttpServlet {
   @EJB // (mapped-name="nomPublic")
   private Hello hello;
   public void service (HttpServletRequest request,
                        HttpServletResponse response)
         throws ServletException, IOException {
      response.getWriter().println(hello.hello());
}  }
```

Installation sous Netbeans :

1. création d'un projet *Java Enterprise Application* :

   `New Project -> Java EE -> Enterprise Application`

2. création d'un *session bean* :

   `New -> Session Bean`

3. sélectionner `Stateless`
4. sélectionner les interfaces souhaitées
5. écrire le code métier dans la classe de bean session

- Les interfaces sont mises à jour par défaut à chaque méthode métier.
- Par défaut, l'objet est publié sous le nom

  `monPaquetage.ClasseduBean`

Exemple de bean session *stateful* avec tags de précompilation (1) :

```
import java.util.*;
import javax.ejb.Stateful; import javax.ejb.Remove;
@Stateful
public class CartBean implements CartRemote, CartLocal {
    String customerName; List<String> contents;

    public void initialize(String person) throws Exception {
        if (person == null) { throw new Exception("Null person.");
        } else { customerName = person; }
        contents = new ArrayList<String>();
    }

    public void addBook(String title) { contents.add(title); }

    public void removeBook(String title) throws Exception {
        boolean result = contents.remove(title);
        if (result == false) {
            throw new Exception(title + " not in cart.");
    }   }

    public List<String> getContents() { return contents; }

    @Remove
    public void remove() { contents = null; }
}
```

Exemple client de bean session *stateful* (2) :

```
import java.util.*;
import javax.naming.*;    import javax.rmi.PortableRemoteObject;
import tools.*;           import <interface>.*;

public class CartClient {
  public static void main(String[] args) {
    try {
      Context initial = new InitialContext();
      Object objref = initial.lookup("java:comp/env/ejb/SimpleCart");

      CartHome home = (CartHome) PortableRemoteObject.narrow
                                          (objref,CartHome.class);
      Cart shoppingCart = home.create("Duke␣DeEarl","123");

      shoppingCart.addBook("The␣Martian␣Chronicles");
      Vector bookList = new Vector();

      bookList = shoppingCart.getContents();
      Enumeration enumer = bookList.elements();
      while (enumer.hasMoreElements()) {
        System.out.println((String) enumer.nextElement()); }

      shoppingCart.removeBook("Alice␣in␣Wonderland");
      shoppingCart.remove();
    } catch (Exception ex) {...;}
} }
```

Exemple de bean session *singleton* :

```
import javax.ejb.Singleton;

// @Startup : démarrage au lancement du conteneur web
@Singleton
// @DependsOn("AutreBeanSingleton")
//    : AUtreBeanSingleton doit être lancé avant ce bean
public class CounterBean {

    private int hits = 1;

    public int getHits() {
        return hits++;
    }
}
```

Beans entité :

- lié à la persistence d'objets/valeurs (cf. Hibernate ...)
- vues d'une base de données
- obligatoirement sérialisables
- possibilité de générer la base à partir de beans entité, ou l'inverse
- identification d'un bean entité par une *clé primaire*
- le type de la clé peut être une classe

La relation bean-BD se fait par un gestionnaire d'entité `EntityManager` qui assure la synchronisation, et (cachée) une *unité de persistence*.

- les appels BD sont définis vis à vis d'un langage spécifique *EJB Query Language*
- nécessite de spécifier les liens entre classes dans un fichier de déploiement ou par des tags : un *schéma abstrait* spécifie les liens entre beans :
  - `One-to-one` : relation biunivoque entre instances de beans
  - `One-to-many`, `Many-to-one`, `Many-to-many` : du même genre

```java
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.*;

@Entity
@NamedQuery(name="findFormation", query="select␣o␣from␣formation␣o")
@Table(name="FORMATION")
@SecondaryTable(name="ETUDIANT")
public class Formation implements Serializable {
   @Id                    // ce qui suit = clé primaire
   @Column(nullable=false)            protected Long id;
   @Column(name="NOM")                protected String nom;
   @Embedded @Column(name="NIVEAU") protected Niveau niveau;
   @OneToMany(mappedBy="formation")
                    protected Collection<Etudiant> etudiants;
   ...
   public Collection<Etudiant> getEtudiants() {
      return etudiants;
   }

   public Etudiant addEtudiant(Etudiant etudiant) {
      getEtudiants().add(etudiant);
      etudiant.setFormation(this);
      return etudiant;
   }
}
```

```
...

@Embeddable
public class Niveau implements Serializable {
   protected String cycle;
   protected String annee;
   public String getAnnee() { return annee; }
   ...
}
```

Il est possible d'associer une hiérarchie d'entités à une ou plusieurs tables :

- `SINGLE_TABLE` : une unique table, un *discriminateur* pour distinguer les sous-classes
- `TABLE_PER_CLASS` : une table par classe concrète
- `JOINED` : une table pour les champs communs, une table pour les champs propres aux sous-classes, opérations implicites de jointure générées

```
@Entity
@Table(name="FORMATION")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn( name="TYPE_FORMATION",
                      discriminatorType=DiscriminatorType.STRING,
                      length=10)
public abstract class Formation implements Serializable { ... }
```

```
@Entity
@DiscriminatorValue(value="alternance")
public class Alternance extends Formation {
   @Column(name="ENTREPRISE")          protected String entreprise;
...
}
```

Les opérations avec la BD se font avec les éléments suivants :

- l'unité de persistence est définie comme une entrée dans un fichier XML `persistence.xml` déclarant les propriétés nécessaires : classes-entités gérées par cette unité, fournisseur du service (i.e. ensemble des classes faisant la gestion concrète de persistence), nom JNDI "de la BD"
- le serveur de persistence fait le lien concret entre le nom JNDI et la BD, définit aussi les informations pour les pools de threads. Le paramétrage de ce serveur peut être donné par une fichier XML. Par exemple :

  `glassfish-resources.xml`

Exemple de fichier `persistence.xml` :

```
<persistence>
   <persistence-unit name="Pizza-ejbPU">
       <description>pour un site de pizza
       </description>
       <jta-data-source>jdbc/PIZZAdb</jta-data-source>
       <jar-file>Pizza.jar</jar-file>
       <class>PizzaManagement</class>
       </persistence-unit>
</persistence>
```

ou

```
<persistence>
  <persistence-unit name="Pizza-ejbPU" transaction-type="JTA">
    <jta-data-source>jdbc/PIZZAdb</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties/>
  </persistence-unit>
</persistence>
```

Exemple de fichier `glassfish-resources.xml` :

```
<resources>
  <jdbc-connection-pool connection-creation-retry-attempts="0"
       connection-validation-method="auto-commit"
       datasource-classname="org.apache.derby.jdbc.ClientDataSource"
       max-pool-size="32"
       max-wait-time-in-millis="60000"
       name="derby_net_PizzaDB_pizzaPool"
       res-type="javax.sql.DataSource"
       steady-pool-size="8">
     <property name="serverName" value="localhost"/>
     <property name="portNumber" value="1527"/>
     <property name="databaseName" value="PizzaDB"/>
     <property name="User" value="pizza"/>
     <property name="Password" value="pizzaUP13"/>
     <property name="URL"
                  value="jdbc:derby://localhost:1527/PizzaDB"/>
     <property name="driverClass"
                  value="org.apache.derby.jdbc.ClientDriver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true"
                   jndi-name="PIZZA" object-type="user"
                   pool-name="derby_net_PizzaDB_pizzaPool"/>
</resources>
```

(possibilité d'administrer “à chaud” via la console d'admin de Glassfish)

- pour les codes utilisant ces classes-entités :
  - un `EntityManager` permet d'effectuer les demandes de lecture/écriture/synchronisation pour/sur des entités
  - cet `EntityManager` peut être à gestion automatique (ex CMP) ou à gestion manuelle (ex BMP) (manuel = création des requêtes, appel explicite de la BD, ...)
  - cet `EntityManager` est paramétré par une *unité de persistence* (via un *contexte de persistence* si gestion automatique).

Exemple :

```
// cas type CMP
public class EtudiantClient {
   @PersistenceContext(unitName="EtudiantUP")
   private EntityManager em;

   public void create(Etudiant etudiant) {
      em.persist(etudiant);
      em.close();
   }
   ...
}
```

```
// cas type BMP
public class EtudiantClient {
   @PersistenceUnit(unitName="EtudiantUP")
   private EntityManagerFactory emf;
   private EntityManager em = emf.createEntityManager();

   public void create(Etudiant etudiant) {
      em.getTransaction().begin();
      em.persist(etudiant);
      em.getTransaction().commit();
   }
   ...
}
```

Entity manager – Opérations de gestion des objets :

`void clear()`	libère tous les objets gérés.
`void persist(Object entity)`	ajoute l'objet pour la persistance.
`void refresh(Object entity)`	actualise l'objet à partir de la BD.
`void detach(Object entity)`	supprime la persistance.
`void flush()`	synchronisation des objets gérés avec la BD.

Entity manager – Requêtes à une BD :

`Query createNamedQuery(String name)`

crée une requête à partir de son alias.

`Query createNativeQuery(String sqlString)`

crée une requête à partir d'une chaîne SQL.

`Query createQuery(String qlString)`

crée une requête à partir d'une chaîne EJBql.

`<T> T find(Class<T> entityClass, Object primaryKey)`

récupère un objet de la BD par sa clé primaire.

(voir exemples de chaînes et opérations sur `Query` dans quelques slides)

Exemple “complet” :

Dans le projet `.war`, les servlets, p.e.,

- `Commande_Pizza.java`

Dans le projet `.ejb`, les beans session et entités, p.e.,

- `PizzaFacade.java` : bean session
- `Pizza.java` : bean entité

```
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;

@Entity
@Table(name = "PIZZA")
@NamedQueries({
  @NamedQuery(name = "Pizza.findAll",
       query = "SELECT p FROM Pizza p"),
  @NamedQuery(name = "Pizza.findByPizzaId",
       query = "SELECT p FROM Pizza p WHERE p.pizzaId = :pizzaId"),
  @NamedQuery(name = "Pizza.findByPrix",
       query = "SELECT p FROM Pizza p WHERE p.prix = :prix")
})
public class Pizza implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "PIZZA_ID")
    private String pizzaId;
    @Column(name = "PRIX")
    private Integer prix;
    @OneToMany(mappedBy = "pizzaId")
    private Collection<Commande> commandeCollection;
    @OneToMany(mappedBy = "pizzaId")
    private Collection<Stock> stockCollection;
... \\suite slide suivant
```

```
...
    public Pizza() { }

    public Pizza(String pizzaId) { this.pizzaId = pizzaId; }

    public String getPizzaId() { return pizzaId; }
    public void setPizzaId(String pizzaId) {
        this.pizzaId = pizzaId;
    }

    public Integer getPrix() { return prix; }
    public void setPrix(Integer prix) { this.prix = prix; }

... // idem autres getters et setters pour les différents champs

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (pizzaId != null ? pizzaId.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // modif similaire à hashCode
    }
}
```

```java
import java.util.List; import javax.ejb.Local;

@Local
public interface PizzaFacadeLocal {
    void create(Pizza pizza);       void edit(Pizza pizza);
    void remove(Pizza pizza);       Pizza find(Object id);
    List<Pizza> findAll();
}
```

```java
import java.util.List; import javax.ejb.Stateless;
import javax.persistence.*;

@Stateless
public class PizzaFacade implements PizzaFacadeLocal {
    @PersistenceContext
    private EntityManager em;

    public void create(Pizza pizza) { em.persist(pizza); }
    public void edit(Pizza pizza) { em.merge(pizza); }
    public void remove(Pizza pizza) { em.remove(em.merge(pizza)); }
    public Pizza find(Object id) { return em.find(Pizza.class,id); }

    public List<Pizza> findAll() {
        return em.createQuery("select␣object(o)␣from␣Pizza␣as␣o")
                 .getResultList();
}   }
```

```
import EntityPizza.CommandeFacadeLocal;
import EntityPizza.Pizza; import EntityPizza.PizzaFacadeLocal;
import EntityPizza.Stock; import EntityPizza.StockFacadeLocal;
import divers.SendMail; import java.io.*; import java.util.*;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Commande_Pizza extends HttpServlet {
    @EJB private CommandeFacadeLocal commandeFacade;
    @EJB private PizzaFacadeLocal pizzaFacade;
    @EJB private StockFacadeLocal stockFacade;

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
    throws ServletException, IOException
        { processRequest(request, response); }

    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
    throws ServletException, IOException
       { processRequest(request, response); }

    @Override
    public String getServletInfo() { return "blabla"; }
... // voir slide suivant
```

```
...
protected void processRequest(HttpServletRequest request,
                              HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Commande␣Pizza</title>");
        out.println("</head>");
        out.println("<body>");
        List lStock = stockFacade.findAll();
        for (Iterator it = lStock.iterator(); it.hasNext();) {
            Pizza elem = ((Stock) it.next()).getPizzaId();
            out.println("Type␣:␣<b>" + elem.getPizzaId() + "␣</b>␣");
            out.println("Prix␣:␣" + elem.getPrix() + "<br/>");
        }
        out.println("<h1>Choisissez␣votre␣pizza␣:␣</h1>");
... // voir slide suivant
```

```
...
    String type=request.getParameter("type");
    if (type!=null) {
      try {
          int quantite=new Integer(request.getParameter("quantite"));
          String email=new String(request.getParameter("email"));
          try {
            Stock s = stockFacade.findByPizzaId(type);
            if (s.getQuantite() > quantite) {
              stockFacade.update(type,s.getQuantite() - quantite);
              Pizza pizza = pizzaFacade.find(type);
              int total = quantite * pizza.getPrix();
              out.println("Commande␣pour␣:"+total+"<br/>");

              commandeFacade.create(type, quantite, total, email);
              out.println("Commande␣effectuée");
              SendMail sM = new SendMail();
              sM.sendMail(email,
                      "Votre␣commande␣de␣pizzas",
                       "Vous␣avez␣commandé␣"
                           + quantite + "␣pizza(s)␣" + type
                       );
            } else {
              out.println("Vous␣demandez␣"+quantite+"␣pizzas␣"
                + "et␣le␣stock␣est␣de␣" + s.getQuantite() + "<br/>");
            };
... // voir slide suivant
```

```
...
            } catch (Exception e) {
                out.println("Nous␣ne␣prenons␣pas␣les␣commandes"
                            + "␣pour␣les␣pizzas␣"+type+"<br/>");
            };
        } catch (Exception ex) { ex.printStackTrace(); }

        } else {
            out.println("<form␣method='POST'>");
            out.println("Type:␣<input␣type='text'"
                  + "␣name='type'><br/>");
            out.println("Quantité:␣<input␣type='text'"
                  + "␣name='quantite'><br/>");
            out.println("Email:␣<input␣type='text'"
                  + "␣name='email'><br/>");
            out.println("<input␣type='submit'><br/>");
            out.println("</form>");
        }
        out.println("</body></html>");
    } finally {
        out.close();
    }
}
}
```

(versions actuelles) Simplification des interfaces pour beans session :

```
import javax.ejb.Stateless;
import javax.persistence.*;

@Stateless
public class PizzaFacade extends AbstractFacade<Pizza>
                         implements PizzaFacadeLocal {
    @PersistenceContext(unitName = "TP4_Pizza-ejbPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public PizzaFacade() {
        super(Pizza.class);
    }

}
```

avec ... (voir slide suivant)

```java
import java.util.List;
import javax.persistence.*;
import javax.persistence.criteria.*;

public abstract class AbstractFacade<T> {
    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager()
                                  .merge(entity));
    }
... // voir slide suivant
```

```
... public T find(Object id) {
      return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
      CriteriaQuery cq = getEntityManager().getCriteriaBuilder()
                                           .createQuery();
      cq.select(cq.from(entityClass));
      return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
      CriteriaQuery cq = getEntityManager().getCriteriaBuilder()
                                           .createQuery();
      cq.select(cq.from(entityClass));
      Query q = getEntityManager().createQuery(cq);
      q.setMaxResults(range[1]-range[0]); q.setFirstResult(range[0]);
      return q.getResultList();
    }

    public int count() {
      CriteriaQuery cq = getEntityManager().getCriteriaBuilder()
                                           .createQuery();
      Root<T> rt = cq.from(entityClass);
      cq.select(getEntityManager().getCriteriaBuilder().count(rt));
      Query q = getEntityManager().createQuery(cq);
      return ((Long) q.getSingleResult()).intValue();
    }
}
```

Exemples de requêtes au format EJBql :

```
SELECT DISTINCT p
FROM Pizza p, IN (p.stock) AS s
WHERE s.quantite = :quantite
```

```
SELECT DISTINCT p
FROM Pizza p
WHERE p.type LIKE 'napo\amp"
```

```
SELECT DISTINCT p
FROM Pizza p
WHERE p.prix BETWEEN :bas AND :haut
```

```
UPDATE Pizza p
SET p.prix = 10
WHERE p.type = 'Napolitaine'
```

```
DELETE
FROM Pizza p
WHERE p.prix BETWEEN :bas AND :haut
```

Pour un environnement de type web + bean session + bean entité (p.e. sous netbeans) :

- on crée les beans entité
- on crée des *facades* pour ces beans entité, i.e. des beans session avec leur interface, intégrant des méthodes par défaut (`create()`, `find()`, ...) (cf menu `New -> Session Beans for Entity classes`)
- on crée par exemple une servlet, ou une JSP, et on injecte le code permettant l'appel d'un bean session (cf menu `Insert Code -> Call Enterprise Bean` ... ou ajout de `@EJB private EtudiantFacadeLocal monEtudiant`)

Gestion de messages (beans messages et paquetage JMS)

- Objets session et entité : synchrones => le client attend
- Objets message :
  - asynchrones => le récepteur peut être en sommeil
  - garantie de livraison (ack)
  - uni ou multicast

Principe :

- un ou plusieurs clients (beans ou non) envoient des messages à une *queue* ou à un *topic*
- un *message-driven bean* extrait de manière asynchrone les messages de la queue ou du topic et les traite
- ou bien des *lecteurs* reçoivent / vont chercher les messages

Au préalable à tout développement de ce type, il faut un **serveur de gestion de messages** intégré à votre serveur d'application :

- soit externe (windows message server, ...)
- soit interne au serveur d'application (dans Sun Java Application Server, Glassfish, ...)

Dans les versions <6 de netbeans, une version téléchargée de Glassfish n'incorpore pas un tel serveur. Il faut l'inclure via les plug-in pour Glassfish.

2 modes sont disponibles :

- mode point à point (`Queue`) : d'un client à un consommateur
- mode publication/abonnement (`Topic`) : d'un client à un ensemble de consommateurs

Un client envoie des messages à une queue d'une fabrique gestionnaire de messages (spécifiée par un nom JNDI).
Le code d'un client doit donc contenir :

- une instance de la classe `ConnectionFactory` créée par un appel au serveur JNDI direct ou par une annotation
- une instance de la classe `Queue` ou `Topic` créée par un appel au serveur JNDI direct ou par une annotation

La création d'une resource (queue ou topic) à déployer dans le serveur de déploiement dépend de ce serveur.

Pour Glassfish :

- les informations se trouvent dans un fichier `glassfish-resources.xml` (dans `Server Resources`),
- création d'une ressource :

  `New -> Glassfish -> JMS Resource`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC
   "-//GlassFish.org//DTD␣GlassFish␣Application␣Server␣3.1␣Resource␣De
   "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <admin-object-resource enabled="true"
              jndi-name="jms/PizzaQueue" object-type="user"
              res-adapter="jmsra" res-type="javax.jms.Queue">
    <description/>
    <property name="Description" value="Pizza"/>
    <property name="Name" value="Pizza"/>
  </admin-object-resource>
</resources>
```

Exemple de producteur de messages :

```
import javax.annotation.Resource;
import javax.jms.*;

public class PizzaProducteur {

  @Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
  private static ConnectionFactory connectionFactory;
  @Resource(lookup = "jms/PizzaQueue")
  private static Queue queue;
// @Resource(lookup = "jms/PizzaTopic")
// private static Topic topic;

  public static void main(String[] args) {
    try (JMSContext context = connectionFactory.createContext();) {
        String message = "Mon message !";
        Destination dest = (Destination) queue;
        context.createProducer().send(dest, message);
    } catch (JMSRuntimeException e) {
        e.printStackTrace();
    }
  }
}
```

Exemple de producteur de messages :

```
import javax.annotation.Resource;
import javax.jms.*;

public class PizzaProducteur {

  @Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
  private static ConnectionFactory connectionFactory;
  @Resource(lookup = "jms/PizzaQueue")
  private static Queue queue;
// @Resource(lookup = "jms/PizzaTopic")
// private static Topic topic;

  public static void main(String[] args) {
    try (JMSContext context = connectionFactory.createContext();) {
        String message = "Mon message !";
        Destination dest = (Destination) queue;
        context.createProducer().send(dest, message);
    } catch (JMSRuntimeException e) {
        e.printStackTrace();
try (..) remplace finally {context.close();}.
  }
}
```

Exemple de consommateur de messages :

```
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup",
                              propertyValue = "jms/PizzaQueue"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue")
})
public class PizzaListener implements MessageListener {

    public PizzaListener() {
    }

    @Override
    public void onMessage(Message message) {
        String s = message.getBody(String.class);
        System.out.println(s);
    }
}
```

Attention ! ! La classe `Message` existe dans `javax.jms` et dans `javax.mail` : elles sont différentes ! ! !

Exemple de consommateur de messages :

```
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup",
                              propertyValue = "jms/PizzaQueue"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue")
})
public class PizzaListener implements MessageListener {
    public PizzaListener() {
    }

    @Override
    public void onMessage(Message message) {
        String s = message.getBody(String.class);
        System.out.println(s);
    }
}
```

Le corps d'un message est une instance de classe et pas nécessairement un texte comme dans JavaMail.

Attention ! ! La classe `Message` existe dans `javax.jms` et dans `javax.mail` : elles sont différentes ! ! !

- Principe : Enterprise Application = ejb + war
- EJB :
  - tables dans une BD
  - Génération de beans **entité** à partir des tables
  - Génération de beans **session** à partir des beans entité
  - Complément avec beans **orientés message**
  - + unité de persistence
- WAR :
  - Servlet (avec appel de beans session)
  - JSP (avec appel de beans session)
  - html

Bean entité

```
@Entity
@Table(name = "STOCK") @XmlRootElement
@NamedQueries({
    @NamedQuery(name="Stock.findStock",
        query="SELECT␣s␣FROM␣Stock␣s␣WHERE␣s.quantite␣=␣:q"),
    @NamedNativeQuery(name="Stock.findQuantite",
        query="SELECT␣s␣FROM␣Stock␣s␣WHERE␣s.quantite␣=␣?")})
public class Stock implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY) @Basic(optional = false)
    @Column(name = "STOCK_ID") private Integer stockId;
    @Column(name = "QUANTITE") private Integer quantite;
    @JoinColumn(name = "PIZZA_ID", referencedColumnName = "PIZZA_ID")
    @ManyToOne private Pizza pizzaId;

    public Stock(Integer stockId) {this.stockId = stockId;}

// et get() et set() ...
}
```

Bean session

```
@Stateless
public class StockFacade extends AbstractFacade<Stock> implements StockFacadeLocal {
    @PersistenceContext(unitName = "TP4_Pizza-ejbPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {return em;}

    public void create(String pizzaId, int quantite) {
        String s = "insert␣into␣stock(pizza_id,quantite)␣";
        s = s + "values␣('" + pizzaId + "'," + quantite + ")";
        em.createNativeQuery(s).executeUpdate();
    }

    pubic List<Stock> list() {
        return em.createNamedQuery("Stock.findStock")
            .setParameter("q",100).getResultList();
    }
    pubic List<Stock> list() {
        return em.createNamedQuery("Stock.findQuantite")
            .setParameter(1,100).getResultList();
    }
    public StockFacade() {super(Stock.class);}
}
```

Entity manager :

`void clear()`	libère tous les objets gérés.
`void persist(Object entity)`	ajoute l'objet pour la persistance.
`void refresh(Object entity)`	actualise l'objet à partir de la BD.
`void detach(Object entity)`	supprime la persistance.
`boolean contains(Object entity)`	teste si l'objet est géré pour la persistance.
`Query createNamedQuery(String name)`	crée une requête à partir de son alias.
`Query createNativeQuery(String sqlString)`	crée une requête à partir d'une chaîne SQL.
`Query createQuery(String qlString)`	crée une requête à partir d'une chaîne EJBql.
`<T> T find(Class<T> entityClass, Object primaryKey)`	récupère un objet de la BD par sa clé primaire.

Les **Web Services** :

- permettent de faire communiquer deux sous-systèmes d'architectures semblables ou différentes de manière standard
- dans le but d'échanger des services ou traitements applicatifs

Deux modèles de web services selon le protocole d'échange :

- via **CorbaRPC** : appels de méthodes à distance après découverte du service (modèle SOAP-RMI)
- via **http simple** : utilisation directe d'URI (modèle REST) par GET/POST

par appel de méthode : le W3C a défini la pile de protocoles

- un service de transport :
  HTTP, *Hypertext Transfer Protocol*
- un système de messages XML :
  SOAP, *Simple Object Access Protocol*
- une description des services :
  WSDL, *Web Services Description Language*
- un système de découverte des services :
  UDDI, *Universal Description Discovery and Integration*

UDDI est un annuaire qui permet de stocker des informations sur des services web.

- Objectif : fournir une architecture de Publication / Découverte / Lien de services web.
- Suppose des standards de description de services, i.e. catégorisation industrielle de services (cf NAICS, UN/SPSC, ...)

Très peu utilisé depuis 2006, reste en intranet

Il peut être présenté en 3 parties :

- **pages blanches** : listent les entreprises ainsi que des informations associées à ces dernières (`Provider`),
- **pages jaunes** : recensent les services web (`Service`) de chacune des entreprises avec le lien vers une implantation du service (`Binding`) sous le standard WSDL,
- **pages vertes** : fournissent des informations techniques précises sur les services fournis (`tModel`).

```
<businessEntity businessKey="...">
  <name>Nom de l'entreprise</name>
  ...
  <businessServices>
   <businessService businessKey="..." serviceKey="...">
     <name>StockQuoteService</name>
     <description xml:lang="fr">...</description>

     <bindingTemplates>
       <bindingTemplate serviceKey="..." bindingKey="...">
         <accessPoint URLType="http">
           http://localhost/HelloWorld/Service1.asmx
         </accessPoint>

         <tModelInstanceDetails>
           <tModelInstanceInfo tModelKey="uuid:..."/>
         </tModelInstanceDetails>

       </bindingTemplate>
     </bindingTemplates>

     <categoryBag>
        <keyedReference
               tModelKey="UUID:..."
               keyName="Stock market trading services"
               keyValue="84121801"/>
      </categoryBag>
   ...
  </businessServices>
  ...
</businessEntity>
```

```
<tModel authorizedName="..." operator="..." tModelKey="...">
   <name>StockQuote Service</name>
   <description xml:lang="en">
      WSDL desc. of a standard stock quote service interface
   </description>

   <overviewDoc>
      <description xml:lang="en">
         WSDL source document
      </description>
      <overviewURL>
         http://stockquote-definitions/stq.wsdl
      </overviewURL>
   </overviewDoc>

   <categoryBag>
      <keyedReference
         tModelKey="UUID:..."
         keyName="uddi-org:types"
         keyValue="wsdlSpec"/>
   </categoryBag>
</tModel>
```

Une requête sur un serveur UDDI est *aussi* un fichier au format XML :

```
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
   <categoryBag>
      <keyedReference
         tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyName="uddi-org:types"
         keyValue="wsdlSpec"/>

      <keyedReference
         tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
         keyName="Stock market trading services"
         keyValue="84121801"/>
   </categoryBag>
</find_tModel>
```

Une requête sur un serveur UDDI est *aussi* un fichier au format XML :

Nom de la requête

```
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
   <categoryBag>
      <keyedReference
         tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyNam
         keyVal
                  
      <keyedRef
         tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
         keyName="Stock market trading services"
         keyValue="84121801"/>
   </categoryBag>
</find_tModel>
```

Spécification du service recherché relativement à une ontologie de services

WSDL, 2 niveaux de description d'un ou plusieurs services dans un même fichier au format XML :

- Abstrait : description générique des services proposés
- Concret : implantation des services décrits abstraitement

WSDL

- Niveau abstrait :
  - Un **type de port** (offre de service) est défini par un ensemble d'opérations
  - Chaque **opération** fait référence à un ou plusieurs messages (i.e. occurrence d'un appel ou d'une réponse)
  - Un **message** est défini par un nom, un ensemble de couples attribut-valeur typés
- Niveau concret :
  - Un **binding** implante un type de port en spécifiant les protocoles concrets utilisés pour chaque opération
  - Un **service** précise pour un binding l'adresse à utiliser

Exemple (1) : Un fichier WSDL

```
<?xml version='1.0' encoding='UTF-8'?>

<definitions
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-...-utility-1.0.xsd"
   xmlns:wsp="http://www.w3.org/ns/ws-policy"
   xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://WS/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http://WS/"
   name="PizzaWSService">

   <wsp:Policy
      xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
      wsu:Id="PizzaWSPortBinding_remove_WSAT_Policy">
      <wsp:ExactlyOne>
         <wsp:All>
            <wsat:ATAlwaysCapability />
            <wsat:ATAssertion
               xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy"
               wsp:Optional="true" ns1:Optional="true" />
         </wsp:All>
      </wsp:ExactlyOne>
   </wsp:Policy>
   <!-- ... pour chacune des opérations -->
...
```

Exemple (1) : Un fichier WSDL

Spécification des espaces de noms nécessaires (chargement pour l'analyse de ce fichier)

Politiques d'utilisation pour une opération (encryptage ou non, horodatage, ...)

```
<?xml version='1.

<definitions
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-...-utility-1.0.xsd"
   xmlns:wsp="http:/
   xmlns:wsp1_2="http                s/2004/09/policy"
   xmlns:wsam="http:/                essing/metadata"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://WS/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http://WS/"
   name="PizzaWSService">

   <wsp:Policy
      xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
      wsu:Id="PizzaWSPortBinding_remove_WSAT_Policy">
      <wsp:ExactlyOne>
         <wsp:All>
            <wsat:ATAlwaysCapability />
            <wsat:ATAssertion
               xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy"
               wsp:Optional="true" ns1:Optional="true" />
         </wsp:All>
      </wsp:ExactlyOne>
   </wsp:Policy>
   <!-- ... pour chacune des opérations -->
...
```

Exemple (1) : Un fichier WSDL

```
<?xml version='1.0' encoding='UTF-8'?>

<definitions
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-...-utility-1.0.xsd"
   xmlns:wsp="http:/
   xmlns:wsp1_2="htt                                    s/2004/09/policy"
   xmlns:wsam="http:/                                   essing/metadata"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://WS/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http
   name="PizzaWSService"

   <wsp:Policy
      xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
      wsu:Id="PizzaWSPortBinding_remove_WSAT_Policy">
      <wsp:ExactlyOne>
         <wsp:All>
            <wsat:ATAlwaysCapability />
            <wsat:ATAssertion
               xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy"
               wsp:Optional="true" ns1:Optional="true" />
         </wsp:All>
      </wsp:ExactlyOne>
   </wsp:Policy>
   <!-- ... pour chacune des opérati
...
```

Politiques d'utilisation pour une opération (encryptage ou non, horodatage, ...)

nom d'identifiant de la politique

le client choisit une politique parmi ce qui suit

le client doit satisfaire à tous les critères dans ce qui suit

indique qu'une transaction atomique débute/finit avec une requête sauf si le client fournit un contexte transactionnel

Exemple (2) :

```
...
   <types>
      <xsd:schema>
         <xsd:import
            namespace="http://WS/"
            schemaLocation="http://localhost:8080/PizzaWSService/PizzaWS?xsd=1" />
      </xsd:schema>
   </types>

   <message name="findAll">
      <part name="parameters" element="tns:findAll" />
   </message>
   <!-- ... pour chacun des messages (i.e. requête de méthode) -->

   <portType name="PizzaWS">
      <operation name="findAll">
         <input
            wsam:Action="http://WS/PizzaWS/findAllRequest"
            message="tns:findAll" />
         <output
            wsam:Action="http://WS/PizzaWS/findAllResponse"
            message="tns:findAllResponse" />
      </operation>
   <!-- ... pour chacune des opérations -->
   </portType>
...
```

Exemple (2)

Définition des types de données utilisées dans les requêtes/réponses

Spécification des types des paramètres nécessaires pour chaque message (i.e. méthode)

Spécification des opérations disponibles

```
...
   <types>
      <xsd:schema>
         <xsd:import
            namespa
            schemaL                    :8080/PizzaWSService/PizzaWS?xsd=1" />
      </xsd:schema>
   </types>

   <message name="findAll">
      <part name="pa                   indAll" />
   </message>
   <!-- ... pour chacun des messages (i.e. requête de méthode) -->

   <portType name="PizzaWS">
      <operation name="findAll">
         <input
            wsam:Action="http://WS/PizzaWS/findAllRequest"
            message="tns:findAll" />
         <output
            wsam:Action="http://WS/PizzaWS/findAllResponse"
            message="tns:findAllResponse" />
      </operation>
   <!-- ... pour chacune des opérations -->
   </portType>
...
```

Exemple (2)

Définition des types de données utilisées dans les requêtes/réponses

URL du schéma XSD définissant les types (cf Exemple (4))

```
...
   <types>
      <xsd:schema>
         <xsd:import
            namespace="http://WS/"
            schemaLocation="http://localhost:8080/PizzaWSService/PizzaWS?xsd=1" />
      </xsd:schema>
   </types>

   <message name="findAll">
      <part name="parameters" element="tns:findAll" />
   </message>
   <!-- ... pour chacun des messages (i.e. requête de méthode) -->

   <portType name="PizzaWS">
      <operation name="findAll">
         <input
            wsam:Action="http://WS/PizzaWS/findAllRequest"
            message="tns:findAll" />
         <output
            wsam:Action="http://WS/PizzaWS/findAllResponse"
            message="tns:findAllResponse" />
      </operation>
   <!-- ... pour chacune des opérations -->
   </portType>
...
```

Exemple (2) :

```
...
   <types>
      <xsd:schema>
         <xsd:import
            namespa
            schemaL                                    :8080/PizzaWSService/PizzaWS?xsd=1" />
      </xsd:schema>
   </types>

   <message name="findAll">
      <part name="parameters" element="tns:findAll" />
   </message>
   <!-- ... pour chacun des messages (i.e. requête de méthode) -->

   <portType name="PizzaWS">
      <operation name="findAll">
         <input
            wsam:Action="http://WS/PizzaWS/findAllRequest"
            message="tns:findAll" />
         <output
            wsam:Action="http://WS/PizzaWS/findAllResponse"
            message="tns:findAllResponse" />
      </operation>
   <!-- ... pour chacune des opérations -->
   </portType>
...
```

Spécification des types des paramètres nécessaires pour chaque message (i.e. méthode)

le type (donné en XML) pour le message findAll

Exemple (2) :

```
...
   <types>
      <xsd:schema>
         <xsd:import
            namespace="http://WS/"
            schemaLocation="http://localhost:8080/PizzaWSService/PizzaWS?xsd=1" />
      </xsd:schema>
   </types>

   <message name="findAll">
      <part name="pa                                    in
   </message>
   <!-- ... pour chacun des messages (i.e. requête de méthode) -->

   <portType name="PizzaWS">
      <operation name="findAll">
         <input
            wsam:Action="http://WS/PizzaWS/findAllRequest"
            message="tns:findAll" />
         <output
            wsam:Action="http://WS/PizzaWS/findAllResponse"
            message="tns:findAllResponse" />
      </operation>
   <!-- ... pour chacune des opérations -->
   </portType>
...
```

Spécification des opérations disponibles

L'opération findAll consiste à :

(1) envoyer une requête à l'URL mentionné, les arguments ont les types indiqués dessous

(2) recevoir une réponse via la requête mentionnée, les arguments ont les types indiqués dessous

Exemple (3) :

```
...
   <binding name="PizzaWSPortBinding" type="tns:PizzaWS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
      <operation name="findAll">
         <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
         <soap:operation soapAction="" />
         <input>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </input>
         <output>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </output>
      </operation>
      <!-- ... binding pour chacun des opérations -->
   </binding>

   <service name="PizzaWSService">
      <port name="PizzaWSPort" binding="tns:PizzaWSPortBinding">
         <soap:address location="http://localhost:8080/PizzaWSService/PizzaWS" />
      </port>
   </service>
</definitions>
```

Exemple (3) : Spécification de l'implantation des opérations

```
...
   <binding name="PizzaWSPortBinding" type="tns:PizzaWS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
      <operation name="findAll">
         <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
         <soap:operation soapAction="" />
         <input>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </input>
         <output>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </output>
      </operation>
      <!-- ... binding operations -->
   </binding>

   <service name="PizzaWSService">
      <port name="PizzaWSPort" binding="tns:PizzaWSPortBinding">
         <soap:address location="http://localhost:8080/PizzaWSService/PizzaWS" />
      </port>
   </service>
</definitions>
```

Spécification de l'implantation des services

Exemple (3) : Spécification de l'implantation des opérations

Protocole de transport utilisé

Politique utilisée

Encodage des entrées

Encodage des sorties

```
...
   <binding name="PizzaWSPortBinding" type="tns:PizzaWS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
      <operation name="findAll">
         <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
         <soap:operation soapAction="" />
         <input>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </input>
         <output>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </output>
      </operation>
      <!-- ... binding pour chacun des opérations -->
   </binding>

   <service name="PizzaWSService">
      <port name="PizzaWSPort" binding="tns:PizzaWSPortBinding">
         <soap:address location="http://localhost:8080/PizzaWSService/PizzaWS" />
      </port>
   </service>
</definitions>
```

Exemple (3) :

```
...
   <binding name="PizzaWSPortBinding" type="tns:PizzaWS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
      <operation name="findAll">
         <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
         <soap:operation soapAction="" />
         <input>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </input>
         <output>
            <wsp:PolicyReference URI="#PizzaWSPortBinding_findAll_WSAT_Policy" />
            <soap:body use="literal" />
         </output>
      </operation>
      <!-- ... bind                                    ati
   </binding>

   <service name="PizzaWSService">
      <port name="PizzaWSPort" binding="tns:PizzaWSPortBinding">
         <soap:address location="http://localhost:8080/PizzaWSService/PizzaWS" />
      </port>
   </service>
</definitions>
```

Spécification de l'implantation des services

lien service abstrait - service concret

adresse du service

Exemple (4) : Schéma XML des types de données pour un service

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:tns="http://WS/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://WS/">

   <xs:element name="findAll" type="tns:findAll" />
   <!-- ... de même pour chaque élément de service -->

   <xs:complexType name="findAll">
      <xs:sequence />
   </xs:complexType>

   <xs:complexType name="findAllResponse">
      <xs:sequence>
         <xs:element name="return" type="tns:stock" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="stock">
      <xs:sequence>
         <xs:element name="pizzaId" type="tns:pizza" minOccurs="0" />
         <xs:element name="quantite" type="xs:int" minOccurs="0" />
         <xs:element name="stockId" type="xs:int" minOccurs="0" />
      </xs:sequence>
   </xs:complexType>

   <!-- ... de même pour chaque type d'arguments -->

</xs:schema>
```

Exemple (4) : Schéma XML des types de données pour un service

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:tns="http://WS/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://WS/">

   <xs:element name="findAll" type="tns:
   <!-- ... de même pour chaque élément de service -->

   <xs:complexType name="findAll">
      <xs:sequence />
   </xs:complexType>

   <xs:complexType name="findAllResponse">
      <xs:sequence>
         <xs:element name="return" type="tns:stock" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="stock">
      <xs:sequence>
         <xs:element name="pizzaId" type="tns:pizza" minOccurs="0" />
         <xs:element name="quantite" type="xs:int" minOccurs="0" />
         <xs:element name="stockId" type="xs:int" minOccurs="0" />
      </xs:sequence>
   </xs:complexType>

   <!-- ... de même pour chaque type d'arguments -->

</xs:schema>
```

le type findAll n'a pas de contenu

le type findAllResponse a un contenu stock

composition d'un stock

Serveur, dans un **ejb** :

- Création d'un Web Service à partir de bean **session** : le bean session servira dans les cas avec états.
- Génération de fichiers wsdl et xsd : serviront pour les clients du web service, ou pour publication.

Client, comme application quelconque (java, C, web, ...) :

- Récupérer des descripteurs wsdl et xsd (ou par code via un serveur de descripteur de web services)
- Le descripteur indique les méthodes utilisables : injecter les appels de ces méthodes dans le code client
- l'exécution de ces méthodes WS passera par : récupération d'un objet `service`, récupération d'un objet `port`, application de la méthode.

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

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

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

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://WSPizza/" name="WebServicePizza"
                                                xmlns:tns="http://WSPizza/">
  <types> <xsd:schema>
   <xsd:import namespace="http://WSPizza/" schemaLocation="WebServicePizza_schema1.xsd"/>
  </xsd:schema> </types>
  <message name="create"> <part name="parameters" element="tns:create"/> </message>
  <portType name="WebServicePizza">
   <operation name="create">
    <input wsam:Action="http://WSPizza/WebServicePizza/create" message="tns:create"/>
   </operation>
  </portType>
  <binding name="WebServicePizzaPortBinding" type="tns:WebServicePizza">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="findAll">
      <soap:operation soapAction=""/>
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>
  </binding>
  <service name="WebServicePizza">
    <port name="WebServicePizzaPort" binding="tns:WebServicePizzaPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port> </service> </definitions>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://WSPizza/" xmlns:tns="http://WSPizza/"
                                                xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="create" type="tns:create"/>
  <xs:complexType name="create">
    <xs:sequence> <xs:element ref="tns:pizza" minOccurs="0"/> </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
public class ClientJavaPizza {
    public static void main(String[] args) {
        .......
    }

    private static Pizza find(java.lang.Object id) {
        WScode.WebServicePizza_Service service = new WScode.WebServicePizza_Service();
        WScode.WebServicePizza port = service.getWebServicePizzaPort();
        return port.find(id);
    }
    private static void create(WScode.Pizza pizza) {
        WScode.WebServicePizza_Service service = new WScode.WebServicePizza_Service();
        WScode.WebServicePizza port = service.getWebServicePizzaPort();
        port.create(pizza);
    }

}
```

```
@WebServiceClient(name = "WebServicePizza", targetNamespace = "http://WSPizza/",
    wsdlLocation = "file:/home/.../jax-ws/resources/WebServicePizza.wsdl")
public class WebServicePizza_Service extends Service
{
    private final static URL WEBSERVICEPIZZA_WSDL_LOCATION;
    private final static WebServiceException WEBSERVICEPIZZA_EXCEPTION;
    private final static QName WEBSERVICEPIZZA_QNAME =
             new QName("http://WSPizza/", "WebServicePizza");

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("file:/home/.../resources/WebServicePizza.wsdl");
        } catch (MalformedURLException ex) {e = new WebServiceException(ex);}
        WEBSERVICEPIZZA_WSDL_LOCATION = url;
        WEBSERVICEPIZZA_EXCEPTION = e;
    }

    public WebServicePizza_Service() {
        super(__getWsdlLocation(), WEBSERVICEPIZZA_QNAME);
    }

    @WebEndpoint(name = "WebServicePizzaPort")
    public WebServicePizza getWebServicePizzaPort() {
        return super.getPort(new QName("http://WSPizza/",
              "WebServicePizzaPort"), WebServicePizza.class, WebServicePizza.class);
    }
}
```

```java
@WebService(name = "WebServicePizza", targetNamespace = "http://WSPizza/")
@XmlSeeAlso({
    ObjectFactory.class
})
public interface WebServicePizza {

    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "find",
        targetNamespace = "http://WSPizza/", className = "WScode.Find")
    @ResponseWrapper(localName = "findResponse",
        targetNamespace = "http://WSPizza/", className = "WScode.FindResponse")
    @Action(input = "http://WSPizza/WebServicePizza/findRequest",
        output = "http://WSPizza/WebServicePizza/findResponse")
    public Pizza find(
        @WebParam(name = "id", targetNamespace = "")
        Object id);

}
```

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "find", propOrder = {
    "id"
})
public class Find {

    protected Object id;
    public Object getId() {
        return id;
    }
    public void setId(Object value) {
        this.id = value;
    }

}
```

REST : *Representational State Transfer* n'est pas un protocole mais un paradigme.

- imaginé par Roy Fielding (année 2000), un des auteurs du protocole HTTP
- Pas uniquement limité aux échanges machines / machines !

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves : a network of web pages (a virtual statemachine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

les URI ont un sens !

- Chaque resource de l'application est accessible via une URI unique
- Les opérations (cf principe CRUD : *Create, Read, Update, Delete*) sont uniformes entre les resources
- Aucune notion d'état dans une suite d'actions (= pas de session utilisateur en HTTP)

On peut imaginer une correspondence entre opérations HTTP et opérations CRUD :

- `GET` correspond quasiment à une opération CRUD Read, toutefois `GET` permet un paramétrage plus fin qu'un simple SELECT.
- `DELETE` correspond à l'opération CRUD Delete
- `PUT` correspond à l'opération CRUD Update, avec les différences suivantes :
  - `PUT` suppose une substitution complète de la ressource, Update peut être partiel.
  - `PUT` peut permettre de créer une ressource (quand l'URI est défini par le client)
- `POST` est utilisé pour une opération CRUD Create. `POST` peut aussi être utilisé pour une mise à jour.

Dans un système REST, les ressources sont manipulées par échange de "représentations" :

- une ressource (un achat par exemple) est représentée par un document XML,
- une opération (par exemple une commande) est effectuée par une requête HTTP (POST avec comme contenu un document XML) envoyé à un URI unique
- l'état de la communication est maintenue à travers la représentation de la ressource

REST :

```
http://gasell.org/users/
http://gasell.org/users/[id] (URI unique)
http://gasell.org/findUser
user = new Resource("http://gasell.org/users/001")
```

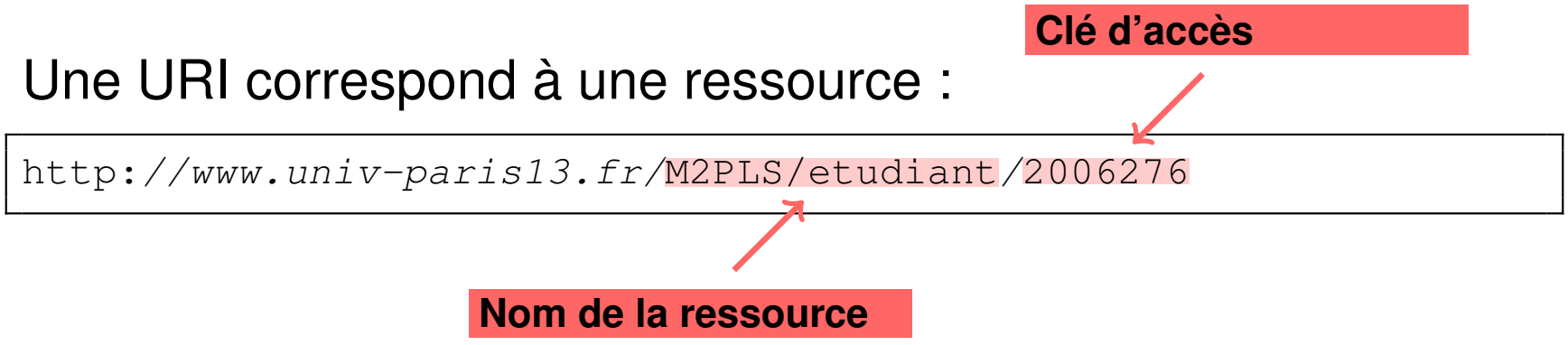
RPC :

```
getUser()
addUser()
removeUser()
updateUser()
findUser()
example = new ExampleApp("gasell.org:1234")
example.getUser()
```

Une URI correspond à une ressource :

```
http://www.univ-paris13.fr/M2PLS/etudiant/2006276
```

Une URI correspond à une ressource :



En résumé :

- les ressources sont identifiées par une URI
- les "méthodes" sont identifiées par une requête http : GET, POST, PUT, DELETE
- une "représentation" est définie par des données et un état transmis via XML

Requête

```
GET /M2PLS/etudiant/2006276 HTTP/1.1
Host: www.univ-paris13.fr
Accept: application/xml
```

Réponse

```
HTTP/1.1 200 OK
Date: Mon, 30 Nov 2009 11:13:43 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8
<?xml version="1.0"?>
<etudiant xmlns="...">
  <personal_data>
    ...
  </personal_data>
  ...
</etudiant>
```

Requête

```
GET /M2PLS/etudiant/2006276 HTTP/1.1
Host: www.univ-paris13.fr
Accept: application/xml
```

Méthode **Ressource**

Réponse

```
HTTP/1.1 200 OK
Date: Mon, 30 Nov 2009 11:13:43 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8
<?xml version="1.0"?>
<etudiant xmlns="...">
  <personal_data>
    ...
  </personal_data>
  ...
</etudiant>
```

Représentation

Cas de J2EE

`JAXRS` : gestion RESTful

`JAXWS` : gestion de web services avec

- `JAXR` publication de services
- `JWSDL` requête de services par web2.0
- `JAXM` gestion des messages
- `JAXRPC` gestion des appels/réponses

Création du service web :

- (Création d'un projet "Web Application" ou "EJB Module")
- Création dans le projet via "New Web Service" (donner un nom de package, e.g. org.up13.eid) ou à partir d'un bean session
- Ajouter des opérations (i.e. services)
- Modifier chaque code (via "Source")
- Run => déploiement et publication du service
- Test possible via le projet de web service

Création d'un client :

- Définir un projet, ou un Web Application, ...
- Créer un fichier par "Web Service Client" en précisant le web service à utiliser [cela charge via wsdl les opérations disponibles]
- Créer une servlet (ou un fichier java, ou une jsp, ...) et faire glisser l'opération souhaitée dans le code source [crée une instance de service, puis de port, puis d'appel d'opération côté client]
- Modifier le code selon les besoins

Serveur, dans un **war** :

- Création d'un Web Service à partir de bean **entity** : REST = gestion sans état !

Client, comme application quelconque (java, C, web, ...) :

- Récupérer des descripteurs wsdl et xsd (ou par code via un serveur de descripteur de web services)
- Le descripteur indique les méthodes utilisables : injecter les appels de ces méthodes dans le code client
- l'exécution de ces méthodes WS passera par : récupération d'un objet `service`, récupération d'un objet `port`, application de la méthode.

```
@Stateless
@Path("entitypizza.pizza")
public class PizzaFacadeREST extends AbstractFacade<Pizza> {
    @PersistenceContext(unitName = "TP4_Pizza-warPU")
    private EntityManager em;

    public PizzaFacadeREST() {  super(Pizza.class); }

    @POST @Override
    @Consumes({"application/xml", "application/json"})
    public void create(Pizza entity) {
        super.create(entity);
    }

    @PUT @Override
    @Consumes({"application/xml", "application/json"})
    public void edit(Pizza entity) {
        super.edit(entity);
    }

    @DELETE
    @Path("{id}")
    public void remove(@PathParam("id") String id) {
        super.remove(super.find(id));
    }

    @GET @Path("{id}")
    @Produces({"application/xml", "application/json"})
    public Pizza find(@PathParam("id") String id) {
        return super.find(id);
    }

    @GET @Override
    @Produces({"application/xml", "application/json"})
    public List<Pizza> findAll() {
        return super.findAll();
    }

    @GET @Path("{from}/{to}")
    @Produces({"application/xml", "application/json"})
    public List<Pizza> findRange(@PathParam("from") Integer from,
        @PathParam("to") Integer to) {
                return super.findRange(new int[]{from, to});
    }

    @Override
    protected EntityManager getEntityManager() { return em; }
}
```

```
public class ClientRESTJavaPizza {

    public static void main(String[] args) {
    PizzaRESTClient client = new PizzaRESTClient();
         Object response = client.count();
         .......
         client.close();
}   }
```

GET http://localhost:19545/TP4_Pizza-war/webresources/entitypizza.pizza/count

```
public class PizzaRESTClient {
    private WebResource webResource;
    private Client client;
    private static final String BASE_URI
             = "http://localhost:19545/TP4_Pizza-war/webresources";

    public PizzaRESTClient() {
        com.sun.jersey.api.client.config.ClientConfig config
             = new com.sun.jersey.api.client.config.DefaultClientConfig();
        client = Client.create(config);
        webResource = client.resource(BASE_URI).path("entitypizza.pizza");
    }

    public String count() throws UniformInterfaceException {
        WebResource resource = webResource;
        resource = resource.path("count");
        return resource.accept(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
    }

    public <T> T findAll_XML(Class<T> responseType) throws UniformInterfaceException {
        WebResource resource = webResource;
        return resource.accept(javax.ws.rs.core.MediaType.APPLICATION_XML)
            .get(responseType);
    }

    public <T> T findRange_XML(Class<T> responseType, String from, String to)
                    throws UniformInterfaceException {
        WebResource resource = webResource;
        resource = resource.path(java.text.MessageFormat.format("{0}/{1}",
                    new Object[]{from, to}));
        return resource.accept(javax.ws.rs.core.MediaType.APPLICATION_XML)
            .get(responseType);
    }
}
```

5 Struts : introduction

- Généralités
- Intercepteurs
- Validation
- Localisation et internationalisation
- Vues : Tiles

- Java
- Opensource, soutenu par l'Apache Software Foundation
- `http://struts.apache.org`
- 2 versions : Struts1 et Struts2
- Nécessite : Servlet API 2.4, JSP API 2.0, Java 5
- Plugins, dont Netbeans
- **Respecte le modèle MVC**

Sous Netbeans : créer un module web en choisissant le framework struts, création d'objets à partir d'un menu spécial struts.

- Contrôleur = servlet
- Vue = JSP (ou Spring, ...)
- Modèle = beans entité (ou Hibernate ...)

Struts parle du contrôleur, donc indépendant de l'implantation pour les vues ou le modèle

Principe concret de fonctionnement :

1. requête d'un browser sur `http://localhost/uneDemande.do`
2. le serveur http (apache, ...) est configuré pour passer toutes les requêtes `*.do` à une instance d'une classe étendant `org.apache.struts.action.ActionServlet` (en regardant le fichier de configuration `web.xml`)

Toutes les demandes du client transitent par le contrôleur :

- C'est une servlet générique fournie par STRUTS.
- Cette servlet générique prend les informations d'initialisation dans le fichier `struts-config.xml`.
- Dans le fichier de configuration `struts-config.html`, à chaque URL devant être traitée :
  - nom d'une *action* : de la sous-classe de la classe `Action` chargée de traiter la requête,
  - nom du *bean* chargé de mémoriser les informations du formulaire si nécessaire.

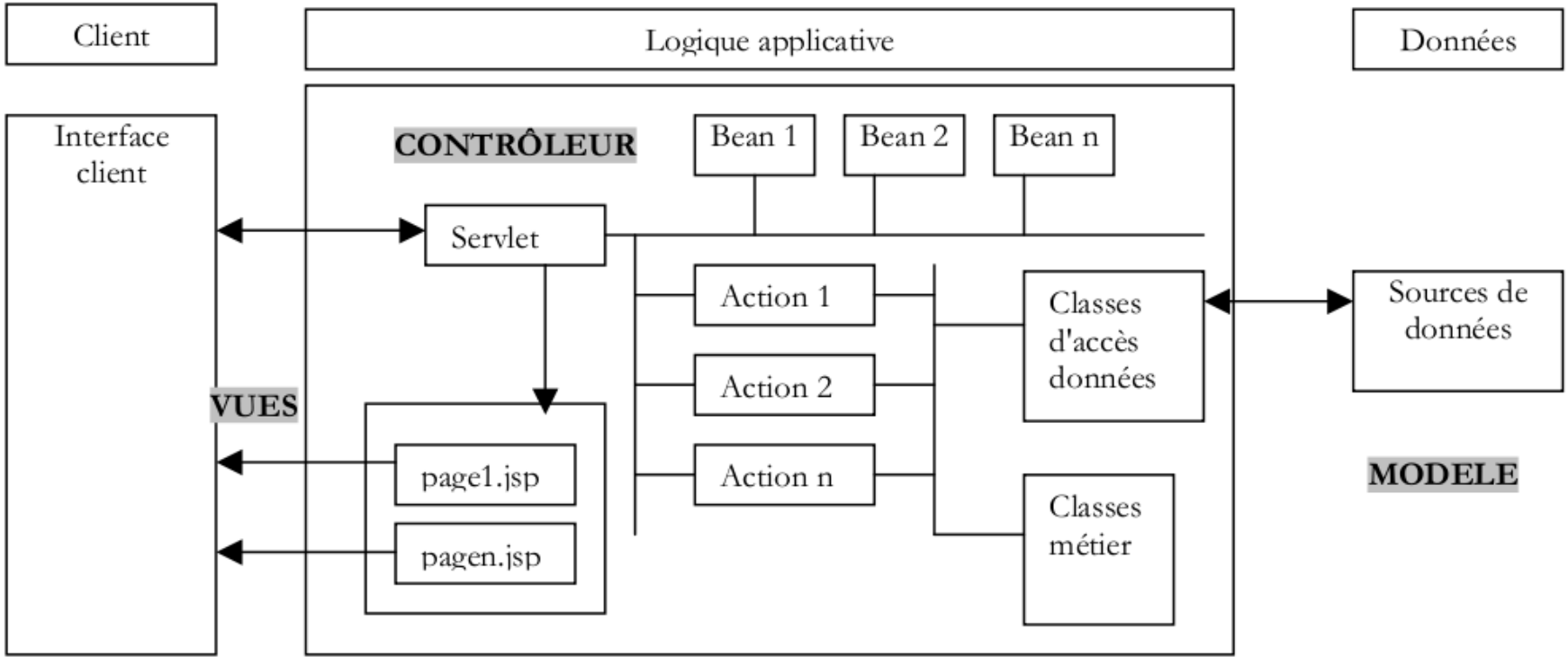
- Si la requête du client contient des paramètres de formulaire, ceux-ci sont mis par le contrôleur dans un objet Bean session, la servlet générique appelle la méthode `validate` du bean session (cette méthode sert à vérifier que les données sont valides ou non).
  - Si les données sont invalides (retour false) alors la servlet générique appelle une vue dont le nom est aussi mentionnée dans `struts-config.html`.
  - sinon la servlet générique appelle la méthode `execute` de l'action associée (en passant comme paramètre la référence au bean session construit).

Action, sous-classe de la classe `Action` :

- la méthode `execute` de l'action contient le code métier correspondant à cette requête,
- à la fin du traitement, l'action rend au contrôleur le nom de la vue qu'il doit envoyer en réponse au client,
- le contrôleur envoie le code généré par la vue au client. L'échange avec le client est terminé.

- Définition des vues :
  - vues formulaire avec informations dans `struts-config.xml` :
    - nom de la classe Bean qui contiendra les données du formulaire, à vérifier ou non,
    - vue à envoyer en réponse au client si données invalides,
    - nom de la classe Action chargée de traiter le formulaire,
    - nom de toutes les vues susceptibles d'être envoyées en réponse au client
  - vues JSP
- Définition des beans correspondant aux vues formulaires.
- Définition des classes Action chargées de traiter les formulaires.
- Définition des classes métier ou autres nécessaires.

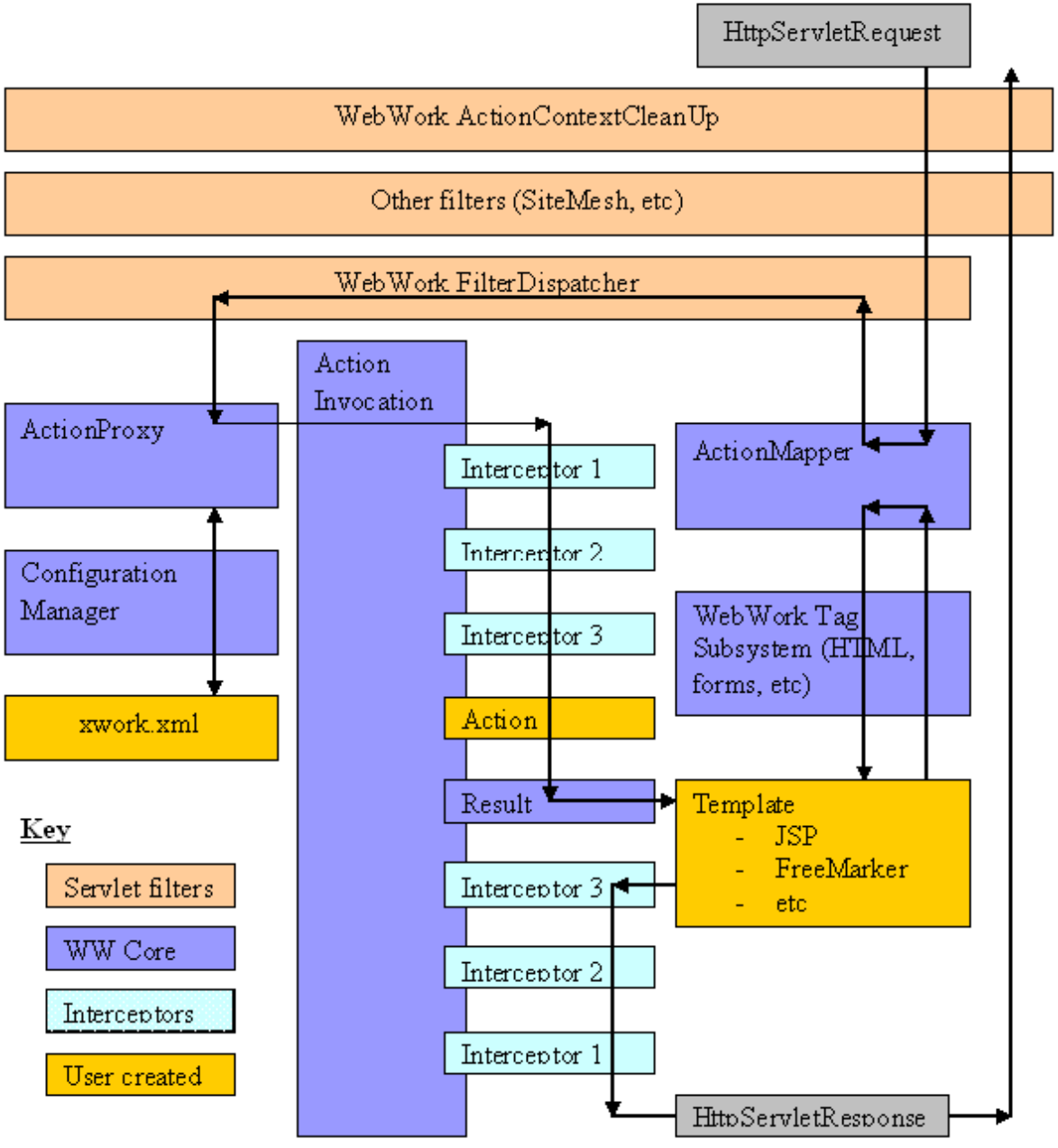
Adaptation du MVC



	Struts 1	**Struts 2**
Action	instance d'une classe abstraite	implém. d'une interface `Action` ou sous-classe de `ActionSupport` ou classe avec `execute()`
`execute()`	lié à `HttpServlet`	map
entrées	sous-classe de `ActionForm`	quelconque
sorties	JSP	ValueStack
`validate()`	objet simple	chaînage sur sous-objets
exécution	via `RequestProcessor`	via piles d'`Interceptor`

Sous netbeans8 : Struts1 en standard, sinon ajouter le plugin ou télécharger les jar de Struts2.

Fonctionnement général



Exemple d'arborescence des sources

- src - java
  - action
    - ClasseAction1.java
    - ClasseAction2.java
  - actionForm
    - ClasseFormAction1.java
    - ClasseFormAction2.java
  - properties
    - ApplicationResource.properties
  - utils
  - datasource
  - databeans

~/web/index.jsp

```
<%@page contentType="text/html; charset=UTF-8" %>
<%@taglib prefix="html" uri="http://struts.apache.org/tags-html"%>

<html>
    <head>
        <title>HELLO</title>
    </head>
    <body>
        <h2>Application HELLO</h2>

        <html:form action="hello" >
            Message: <html:text property="message" />
            <html:submit />
        </html:form>

    </body>
</html>
```

Exemple STRUTS1 : Ecriture de la vue "Succes"

~/web/succes.jsp

```
<%@page contentType="text/html;␣charset=UTF-8" %>
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
    <head>
        <title>HELLO</title>
    </head>
    <body>
        <h2>Application HELLO</h2>
        <h3><bean:write name="helloForm" property="message" /></h3>
    </body>
</html>
```

~/web/erreur.jsp

```
<%@page contentType="text/html;␣charset=UTF-8" %>
<html>
    <head>
        <title>HELLO</title>
    </head>
    <body>
        <h1>Erreur</h1>
        Il fallait entrer le message "bonjour"
    </body>
</html>
```

~/src/java/dev/monWebStruts/HelloForm.java

```
package dev.monWebStruts;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.*;

public class HelloForm extends org.apache.struts.action.ActionForm {

    private String message;
    public String getMessage() {return message;}
    public void setMessage(String message) {this.message = message;}
    public HelloForm() {super();}

    /**
     * @param mapping The ActionMapping used to select this instance.
     * @param request The HTTP Request we are processing.
     */
    public ActionErrors validate(ActionMapping mapping,
                                 HttpServletRequest request) {
        ActionErrors errors = new ActionErrors();
        if (getMessage() == null || getMessage().length() < 1) {
            errors.add("message",
                       new ActionMessage("error.name.required"));
        }
        return errors;
    }
}
```

~/src/java/dev/monWebStruts/Hello.java

```
package dev.monWebStruts;
import javax.servlet.http.*;    import org.apache.struts.action.*;

public class Hello extends org.apache.struts.action.Action {

    /**
     * @param mapping The ActionMapping used to select this instance.
     * @param form The optional ActionForm bean for this request.
     * @param request The HTTP Request we are processing.
     * @param response The HTTP Response we are processing.
     */
    @Override
    public ActionForward execute(
            ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        String message = ((HelloForm) form).getMessage();
        if (message.equalsIgnoreCase("bonjour")) {
        return mapping.findForward("succes");
        } else {
        return mapping.findForward("erreur");
        }
    }
}
```

Exemple STRUTS1 : Configuration du contrôleur

~/web/WEB-INF/struts-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<struts-config>
    <form-beans>
        <form-bean name="helloForm"
                   type="dev.monWebStruts.HelloForm"/>
    </form-beans>

    <action-mappings>
        <action path="/hello"
                type="dev.monWebStruts.Hello"
                name="helloForm"
                input="/index.jsp"
                validate="true">
            <forward name="succes" path="/succes.jsp"/>
            <forward name="erreur" path="/erreur.jsp"/>
        </action>
    </action-mappings>

    <message-resources
                parameter="dev/monWebStruts/ApplicationResource"/>

</struts-config>
```

~/src/java/dev/monWebStruts/ApplicationResource.properties

```
error.name.required=<b>Entrez au moins 1 caractère !</b>
# -- standard errors --
errors.header=<UL>
errors.prefix=<LI>
errors.suffix=</LI>
errors.footer=</UL>
# -- validator --
errors.invalid={0} is invalid.
errors.maxlength={0} can not be greater than {1} characters.
errors.minlength={0} can not be less than {1} characters.
...
```

~/web/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
            org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

~/web/index.jsp

```
<%@page contentType="text/html; charset=UTF-8" %>
<%@taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>HELLO</title>
    </head>
    <body>
        <h2>Application HELLO</h2>

        <s:form action="hello" >
            <s:textfield name="message" label="Message" />
            <s:submit />
        </s:form>

    </body>
</html>
```

Exemple STRUTS2 : Ecriture de la vue "Succes"

~/web/succes.jsp

```
<%@page contentType="text/html; charset=UTF-8" %>
<%@taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>HELLO</title>
    </head>
    <body>
        <h2>Application HELLO</h2>
        <h3><s:property value="message" /></h3>
    </body>
</html>
```

Exemple STRUTS2 : Ecriture de la vue "Erreur"

~/web/erreur.jsp

```
<%@page contentType="text/html; charset=UTF-8" %>
<%@taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>HELLO</title>
    </head>
    <body>
        <h1>Erreur</h1>
        Il fallait entrer le message "bonjour"
    </body>
</html>
```

~/src/java/action/Hello.java

```
package action;
import com.opensymphony.xwork2.ActionSupport;

public class Hello extends ActionSupport {

    private String message;

    public String execute() throws Exception {
        if (message.equalsIgnoreCase("bonjour")) {
            return "succes";
        } else {
            return "erreur";
        }
    }

    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

~/src/java/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="test" extends="struts-default">
    <action name="hello" class="action.Hello" method="execute">
       <result name="succes">succes.jsp</result>
       <result name="erreur">erreur.jsp</result>
       <result name="input">index.jsp</result>
    </action>
  </package>
</struts>
```

Dans une page JSP, le développeur peut :

- récupérer des objets dans la requête (`request.getAttribute(...)`), la session (`session.getAttribute(...)`, ou le contexte de l'application
- inclure des parties dynamiques dans le code HTML au moyen de variables `<\%= variable \%>`
- injecter du code Java `<\% code Java \%>`

STRUTS ajoute des bibliothèques de balises (taglib) :

- `struts-html`
- `struts-logic` (logique d'exécution)
- `struts-bean` (accès aux objets contenus dans la requête, la session, le contexte)

- permet d'éviter le code Java dans les JSP !
- permet d'afficher des données dynamiques :

```
<s:textfield name="postalCode"/>
```

- permet d'afficher un label avec le champs input :

```
<s:textfield key="postalCode.label" name="postalCode"/>
```

- Notez que les valeurs sont des objets :

```
<s:textfield key="pays.label" name="pays" value="%{'FRANCE'}" />
```

- HTML : formulaires
- BEAN : instancier une classe dans une JSP
- LOGIC : tests, itérations...

Pour Struts2, adresse de documentation :

`struts.apache.org/release/2.0.x/docs/tag-reference.html`

Mise en place :

`index.jsp` :

```
<%@taglib prefix="html" uri="/struts-tags" %>
<html:html>
...
</html:html>
```

`web.xml` :

```
  <taglib>
    <taglib-uri>StrutsHtml</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
```

~src/java/model/Convertisseur.java

```
package model;
public class Convertisseur {

    private double francs;
    private double euros;

    public double getEuros() {
        return francs / 6.55957;
    }
    public void setEuros(double euros) {
        this.euros = euros;
    }
    public double getFrancs() {
        return euros * 6.55957;
    }
    public void setFrancs(double francs) {
        this.francs = francs;
    }
}
```

~/web/convertisseur.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=ISO-8859-1">
        <title>CONVERTISSEUR</title>
    </head>
    <body>
      <h1>Convertisseur de devise</h1>
      <s:bean name="model.Convertisseur" var="converter1">
          <s:param name="euros" value="100"></s:param>
      </s:bean>
      <s:bean name="model.Convertisseur" var="converter2">
          <s:param name="francs" value="100"></s:param>
      </s:bean>
      100 francs = <s:property value="#converter2.euros" /> euros
      <br>
      100 euros = <s:property value="#converter1.francs" /> francs
    </body>
</html>
```

~/src/java/model/IndividuModel.java

```
package model;

public class IndividuModel {
    String identifiant;
    String courriel;
    public IndividuModel(){}
    public IndividuModel(String identifiant,String courriel){
        this.identifiant=identifiant;
        this.courriel=courriel;
    }
    // Accesseurs...
}
```

~/src/java/action/IteratorAction.java

```
package action;

import com.opensymphony.xwork2.ActionSupport;
import java.util.ArrayList;
import java.util.List;
import model.IndividuModel;

public class IteratorAction extends ActionSupport {
    private List individusList;

    public String execute() {
        individusList = new ArrayList<IndividuModel>();
        individusList.add(new IndividuModel("michael",
                                            "michael@lipn.fr"));
        individusList.add(new IndividuModel("christophe",
                                            "christophe@lipn.fr"));
        return "succes";
    }
    public List getIndividusList() { return individusList; }
    public void setIndividusList(List individusList) {
        this.individusList = individusList;
    }
}
```

~/src/java/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="test" extends="struts-default">
   ...
       <action name="iteratorTag" class="action.IteratorAction">
            <result name="succes">/tag_logic.jsp</result>
       </action>
   ...
  </package>
</struts>
```

~/web/tag_logic.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=ISO-8859-1">
        <title>TAG LOGIC</title>
    </head>
    <body>
        <h1>TAG LOGIC</h1>
        <s:iterator value="individusList">
            <s:property value="identifiant"/>
            : <s:property value="courriel"/><br>
        </s:iterator>
    </body>
</html>
```

Champs de formulaires

- textfield
- password
- radio
- select
- textarea
- checkboxlist
- checkbox
- submit

~/web/enregistrement.jsp

```
<s:textfield name="nom" label="Nom" value=""/>
```

~/src/java/action/EnregistrementAction.java

```
...
    private String nom;
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
...
```

~/web/enregistrement.jsp

```
<s:password name="password" label="Mot de passe" />
```

~/src/java/action/EnregistrementAction.java

```
...
    private String mdp;
    public String getMdp() {
        return mdp;
    }
    public void setMdp(String mdp) {
        this.mdp = mdp;
    }
...
```

~/web/enregistrement.jsp

```
    <s:radio name="sexe" label="Sexe" list="{'Homme','Femme'}" />
```

~/src/java/action/EnregistrementAction.java

```
    private String sexe;
    public String getSexe() {
        return sexe;
    }
    public void setSexe(String sexe) {
        this.sexe = sexe;
    }
```

~/web/enregistrement.jsp

```
<s:select name="pays" list="paysList" listKey="id"
          listValue="nom" headerKey="0"
          headerValue="Pays" label="Choisissez un pays" />
```

~/src/java/action/EnregistrementAction.java

```
private String pays;
private ArrayList<Pays> paysList;
public String getPays() {
   return pays;
}
public void setPays(String pays) {
   this.pays = pays;
}
public ArrayList<Pays> getPaysList() {
   return paysList;
}
public void setPaysList(ArrayList<Pays> paysList) {
   this.paysList = paysList;
}
```

~/web/enregistrement.jsp

```
    <s:textarea name="remarque" label="Remarque" />
```

~/src/java/action/EnregistrementAction.java

```
    private String remarque;
    public String getRemarque() {
        return remarque;
    }
    public void setRemarque(String remarque) {
        this.remarque = remarque;
    }
```

~/web/enregistrement.jsp

```
<s:checkboxlist list="communauteList" name="communaute"
                                      label="Communaute" />
```

~/src/java/action/EnregistrementAction.java

```
    private String[] communaute;
    private ArrayList<String> communauteList;
    public String[] getCommunaute() {
        return communaute;
    }
    public void setCommunaute(String[] communaute) {
        this.communaute = communaute;
    }
    public ArrayList<String> getCommunauteList() {
        return communauteList;
    }
    public void setCommunauteList(ArrayList<String> communauteList) {
        this.communauteList = communauteList;
    }
```

~/web/enregistrement.jsp

```
    <s:checkbox name="mailingList"
                label="Abonnement␣a␣la␣liste␣de␣diffusion␣?" />
```

~/src/java/action/EnregistrementAction.java

```
    private Boolean  mailingList;
    public Boolean getMailingList() {
        return mailingList;
    }
    public void setMailingList(Boolean mailingList) {
        this.mailingList = mailingList;
    }
```

~/web/enregistrement.jsp

```
    <s:submit value="Enregistrer" />
```

~/src/java/action/EnregistrementAction.java

```
    public String populate() {
        paysList = new ArrayList<Pays>();
        paysList.add(new Pays(1, "Allemagne"));
        paysList.add(new Pays(2, "USA"));
        paysList.add(new Pays(3, "France"));

        communauteList = new ArrayList<String>();
        communauteList.add("Java");
        communauteList.add(".Net");
        communauteList.add("Linux");
        communaute = new String[]{"Java",".Net"};

        mailingList = true;

        return "populate";
    }
```

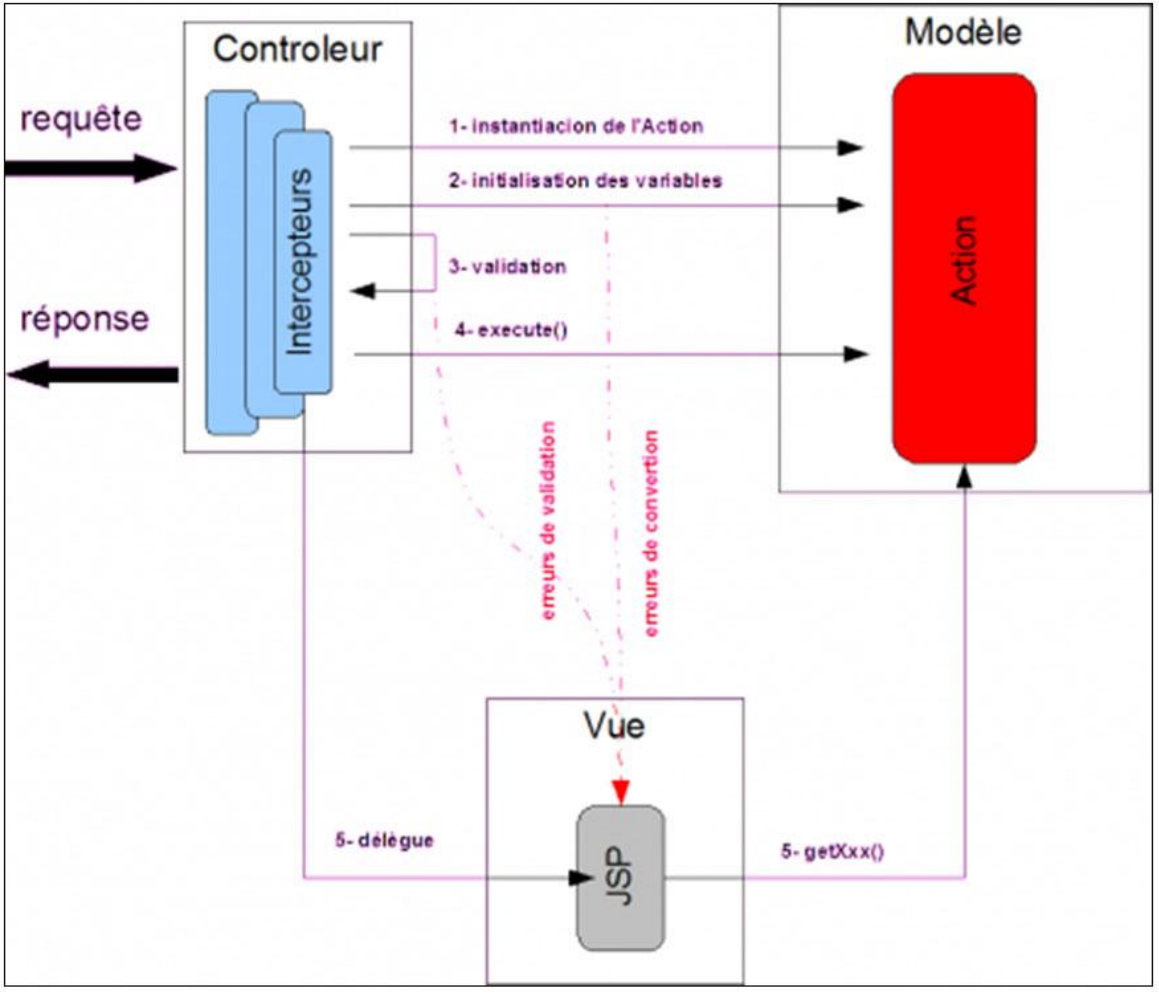
~/src/java/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
     "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
     "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="test" extends="struts-default">
   ...
       <action name="*Enregistrement" method="{1}"
               class="action.EnregistrementAction">
           <result name="populate">/enregistrement.jsp</result>
           <result name="input">/enregistrement.jsp</result>
           <result name="succes">/enregistrement_succes.jsp</result>
       </action>
   ...
  </package>
</struts>
```

~/web/index.jsp

```
...
<li>Bean: <a href="http://localhost:8080/TAGS/convertisseur.jsp">
          Convertisseur
          </a></li>

<li>Logic: <a href="<s:url␣action='iteratorTag'/>">
          Iterator
          </a></li>
   <s:url action="populateEnregistrement" var="enregistrement">
      <s:param name="titre">Formulaire</s:param>
   </s:url>

<li>Html: <a href="${enregistrement}">
          Enregistrement
          </a></li>
...
```

Workflow global



- Flexibilité : on ajoute les intercepteurs où c'est nécessaire
- Propreté/lisibilité du code $\rightarrow$ on se focalise sur l'action
- Réutilisabilité
- Tests facilités

Exemple : journalisation (logging)

~/src/java/intercepteur/Logging.java

```
package intercepteur;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;

public class Logging implements Interceptor{
   public String intercept(ActionInvocation invocation)
           throws Exception {
      String className = invocation.getAction().getClass().getName();
      long startTime = System.currentTimeMillis();
      System.out.println("Appel de l'action: " + className);

      String result = invocation.invoke();

      long endTime = System.currentTimeMillis();
      System.out.println("Duree: "+ (endTime-startTime) +" ms");

      return result;
   }

   public void destroy() { // Destruction de l'intercepteur
   }
   public void init() { // Initialisation de l'intercepteur
}  }
```

~/src/java/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="default" extends="struts-default">
     ...
     <interceptors>
        <interceptor name="mylogging" class="intercepteur.Logging" />
        <interceptor-stack name="loggingStack">
          <interceptor-ref name="mylogging" />
          <interceptor-ref name="defaultStack" />
        </interceptor-stack>
     </interceptors>
     ...
```

~/src/java/struts.xml

```
      ...
      <action name="LoginIntercept" class="action.Login">
         <interceptor-ref name="loggingStack"></interceptor-ref>
         <result name="input">/login.jsp</result>
         <result name="success">/accueil.jsp</result>
       </action>
    ...
  </package>
</struts>
```

~/src/java/struts.xml

```
<interceptor-stack name="defaultStack">
     <interceptor-ref name="exception"/>
     <interceptor-ref name="servletConfig"/>
     <interceptor-ref name="prepare"/>
     <interceptor-ref name="i18n"/>
     <interceptor-ref name="chain"/>
     <interceptor-ref name="debugging"/>
     <interceptor-ref name="profiling"/>
     <interceptor-ref name="staticParams"/>
     <interceptor-ref name="actionMappingParams"/>
     <interceptor-ref name="params">
        <param name="excludeParams">dojo\..*,^struts\..*</param>
     </interceptor-ref>
     <interceptor-ref name="conversionError"/>
     <interceptor-ref name="validation">
        <param name="excludeMethods">input,back,cancel,browse</param>
     </interceptor-ref>
     <interceptor-ref name="workflow">
        <param name="excludeMethods">input,back,cancel,browse</param>
     </interceptor-ref>
 </interceptor-stack>
```

- Contrôle de surface (qualitative)
  - Vérifier que les données saisies sont bien dans la forme attendue
    Ex : une donnée numérique ne contient que des chiffres
- Validation sémantique
  - Vérifier que la valeur saisie est bien celle qui est attendue par le système
    Ex : un numéro de carte bleue valide

- soit méthode `validate()` d'une ActionForm
- soit utilisation d'un fichier XML de la forme ActionClass-validation.xml

`http://struts.apache.org/2.3.8/docs/validation.html`

- required
- requiredstring
- int
- long
- short
- double
- date
- expression
- fieldexpression
- email
- url
- visitor
- conversion
- stringlength
- regex
- conditionalvisitor

- public static final String ERROR = "error"
- public static final String INPUT = "input"
- public static final String LOGIN = "login"
- public static final String NONE = "none"
- public static final String SUCCESS = "success"

~/web/login.jsp

```
<%@taglib uri="/struts-tags" prefix="s" %>
<html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Page de connexion</title>
    <s:head />
  </head>
  <body>
    <s:form action="Login">
        <s:textfield name="identifiant" label="Identifiant" />
        <s:password name="mdp" label="Mot de passe" />
        <s:submit value="Se connecter" />
    </s:form>
  </body>
</html>
```

~/src/java/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="default" extends="struts-default">
   ...
       <action name="Login" class="action.Login">
            <result name="input">/login.jsp</result>
            <result name="success">/accueil.jsp</result>
        </action>
   ...
  </package>
</struts>
```

Exemple 1 : validation avec ActionForm → Action

~/src/java/action/Login.java

```
package action;
import com.opensymphony.xwork2.ActionSupport;

public class Login extends ActionSupport {
    private String identifiant;
    private String mdp;

    public String execute() { return SUCCESS; }

    public void validate() {
      if (getIdentifiant().length() == 0) {
          addFieldError("identifiant", "L'identifiant est requis");
      } else if (!getIdentifiant().equals("michael")) {
          addFieldError("identifiant", "Utilisateur non autorise");
      }
      if (getMdp().length() == 0) {
          addFieldError("mdp", "le mot de passe est obligatoire");
      }
    }

    public Login() {}
    // ... accesseurs
}
```

~/src/java/action/Login-validation.xml

```
<!DOCTYPE validators PUBLIC
  "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
  "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
    <field name="identifiant">
        <field-validator type="requiredstring">
         <message>Identifiant obligatoire !</message>
        </field-validator>
    </field>
    <field name="mdp">
        <field-validator type="requiredstring">
         <message>Mot de passe obligatoire !</message>
        </field-validator>
    </field>
</validators>
```

Tous les messages textuels doivent être dans des fichiers séparés du code source (Java, HTML...) $\rightarrow$ Resources Bundle

- Struts : plusieurs niveaux
  - Application
  - Package
  - Action
- Fichiers *.properties : messages.properties
- Internationalisation en suffixant le code pays :
  - francais : messages_fr.properties
  - anglais : messages_en.properties
  - italien : messages_it.properties
  - ...

~/src/java/action/messages.properties

```
titre=LOCALISATION
bienvenue=Bienvenue
page.langue=La page que vous visualisez est en francais
```

~/src/java/action/messages.properties_fr

```
titre=LOCALISATION
bienvenue=Bienvenue
page.langue=La page que vous visualisez est en francais
```

~/src/java/action/messages.properties_en

```
titre=LOCALIZATION
bienvenue=Welcome
page.langue=This page is in english
```

~/src/java/action/**package**.properties

```
action=Une action est intervenue
```

~/src/java/action/**package**.properties_fr

```
action=Une action est intervenue
```

~/src/java/action/**package**.properties_en

```
action=An action was intercepted
```

~/src/java/action/LocaleAction.properties

```
accueil=Vous etes sur la page d'accueil
#␣Formulaire␣de␣changement␣de␣langue
langue_disponible=Langues␣disponibles
francais=Francais
anglais=Anglais
```

~/src/java/action/LocaleAction.properties_fr

```
accueil=Vous etes sur la page d'accueil
#␣Formulaire␣de␣changement␣de␣langue
langue_disponible=Langues␣disponibles
francais=Francais
anglais=Anglais
```

~/src/java/action/LocaleAction.properties_en

```
accueil=You are on the Welcome Page
# Formulaire de changement de langue
langue_disponible=Available Languages
francais=French
anglais=English
```

- Niveau application : `~/src/java/struts.xml`

```
<struts>
 <constant  name="struts.custom.i18n.resources" value="messages" />

 <package name="action" extends="struts-default">
    <action name="change_langue" method="execute"
            class="action.LocaleAction" >
        <result name="change">/accueil.jsp</result>
    </action>
    <action name="traduit" method="traduit"
            class="action.TraduitAction" >
        <result name="traduit">/accueil.jsp</result>
    </action>
    <action name="accueil" method="accueil"
            class="action.AccueilAction">
        <interceptor-ref name="i18n"/>
        <result name="accueil">/accueil.jsp</result>
    </action>
 </package>
</struts>
```

- Niveau Package : package.properties (+suffix pays)
- Niveau Action : nom_classe_action.properties (+suffix pays)

~/web/accueil.jsp

```
<s:property value="getText('bienvenue')" />

<s:text name="bienvenue" />

<s:text name="test-test" >
   Ce message s'affiche car la cle "test-test" n'existe pas dans
   les fichiers de langue
</s:text>

<s:form action="traduit" >
   <s:textfield name="cle" key="cle" />
   <s:submit key="submit" />
</s:form>
```

~/src/java/action/TraduitAction.java

```
package action;
import com.opensymphony.xwork2.ActionSupport;

public class TraduitAction extends ActionSupport {
    private String cle;
    private String traduction;

    public String traduit() {
        if (cle != null) {
            String str_traduction = getText(cle);

            if (!str_traduction.equalsIgnoreCase(cle)) {
                this.setTraduction(str_traduction);
            } else {
                this.setTraduction("Cle inexistante");
            }
        } else {
            this.setTraduction("Cle inexistante");
        }
        return "traduit";
    }

    // ... accesseurs
}
```

Exemple : Changer de langue (1)

~/web/accueil.jsp

```
<!-- Creation des liens -->
<s:url id="localeFR" action="change_langue">
   <s:param name="request_locale">fr</s:param>
</s:url>

<s:url id="localeEN" action="change_langue">
   <s:param name="request_locale">en</s:param>
</s:url>


<!-- Affichage des liens -->
<s:a href="%{localeFR}"><s:text name="francais" /></s:a>
<s:a href="%{localeEN}"><s:text name="anglais" /></s:a>
```

→ http://localhost:8080/accueil.action?request_locale=fr

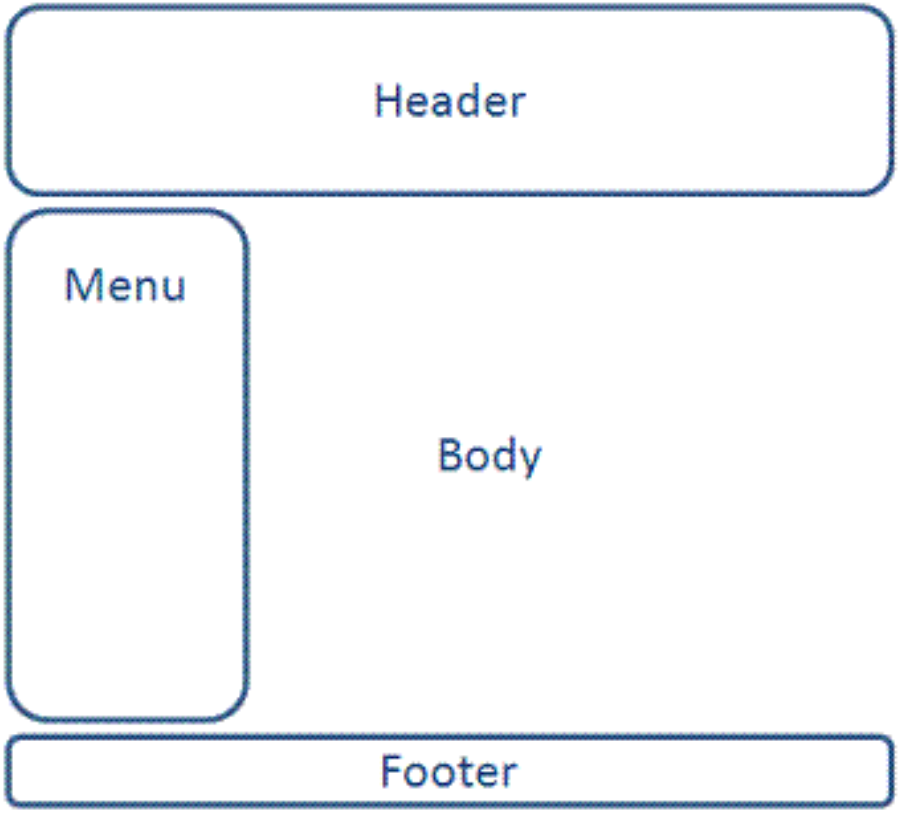
~/src/java/action/LocaleAction.java

```
package action;
import com.opensymphony.xwork2.ActionSupport;

public class LocaleAction extends ActionSupport {
    public String execute() throws Exception {
        return "change";
    }
}
```

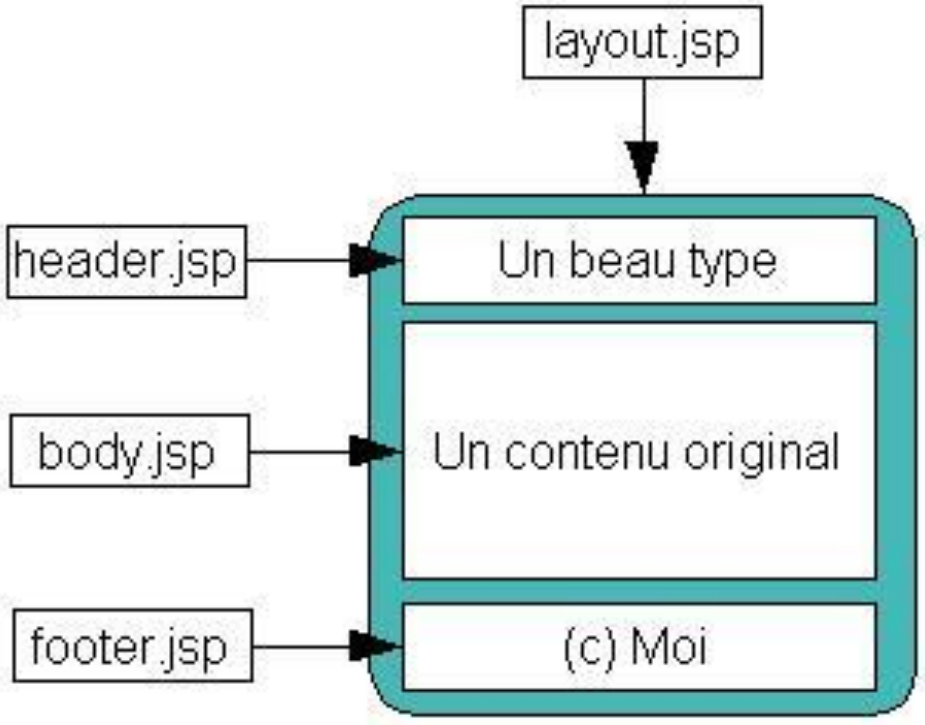
~/src/java/struts.xml

```
<action name="change_langue" method="execute"
                                class="action.LocaleAction" >
    <result name="change">/accueil.jsp</result>
</action>
```

Disposition (layout)



→ Templates



→ Fournit l'apparence de la page

- Système de template
- Construction et chargement de pages JSP dynamiques
- Supporte la réutilisation de "tile"
- Supporte l'internationnalisation (I18N)
- Intégration dans Struts via plugin

→ `http://tiles.apache.org`

~/web/WEB-INF/web.xml

```
<context-param>
    <param-name>tilesDefinitions</param-name>
    <param-value>/WEB-INF/tiles.xml</param-value>
</context-param>

<listener>
   <listener-class>
        org.apache.struts2.tiles.StrutsTilesListener
   </listener-class>
</listener>
```

~/web/WEB-INF/tiles.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
   "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
   "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">

<tiles-definitions>
   <definition name="baseLayout" template="/layout/base.jsp">
      <put-attribute name="title" value="" />
      <put-attribute name="header" value="/templates/header.jsp" />
      <put-attribute name="menu" value="/templates/menu.jsp" />
      <put-attribute name="body" value="" />
      <put-attribute name="footer" value="/templates/footer.jsp" />
   </definition>
...
```

~/web/WEB-INF/tiles.xml

```
...
   <definition name="accueil" extends="baseLayout">
      <put-attribute name="title" value="Accueil" />
      <put-attribute name="body" value="/pages/accueil.jsp" />
   </definition>
   <definition name="about" extends="baseLayout">
      <put-attribute name="title" value="A␣propos␣de..." />
      <put-attribute name="body" value="/pages/about.jsp" />
   </definition>
   <definition name="contact" extends="baseLayout">
      <put-attribute name="title" value="Contact" />
      <put-attribute name="body" value="/pages/contact.jsp" />
   </definition>
</tiles-definitions>
```

~/web/layout/base.jsp

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD␣HTML␣4.01␣Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
   <meta http-equiv="Content-Type" content="text/html;␣charset=UTF-8">
      <title>
         <tiles:insertAttribute name="title" ignore="true" />
      </title>
 </head>
 <body>
   <table border="1" align="center" width="100%">
      <tr>
         <td><tiles:insertAttribute name="header" /></td>
      </tr>
      <tr>
         <td><tiles:insertAttribute name="menu" /></td>
         <td><tiles:insertAttribute name="body" /></td>
      </tr>
      <tr>
         <td><tiles:insertAttribute name="footer" /></td>
      </tr>
   </table>
 </body></html>
```

~/web/templates/header.jsp

```
<%@ page contentType="text/html;␣charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<h2 align="center">Header</h2>
```

~/web/templates/body.jsp

```
<%@ page contentType="text/html;␣charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
Body
```

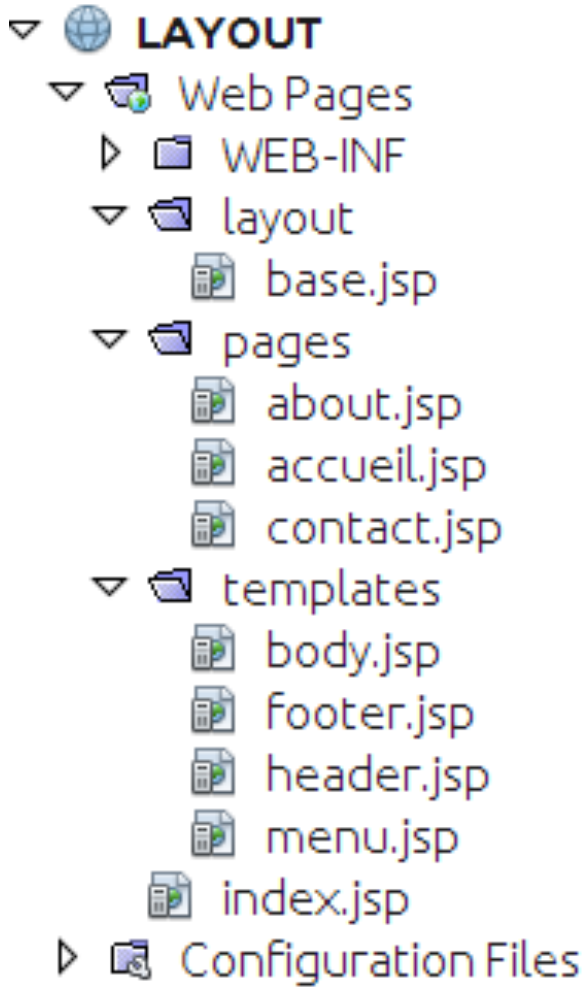
~/web/templates/footer.jsp

```
<%@ page contentType="text/html;␣charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<div align="center">Footer</div>
```

~/web/templates/menu.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<ul>
    <li><a href="<s:url action="accueil"/>">Accueil</a></li>
    <li><a href="<s:url action="about"/>">A propos de...</a></li>
    <li><a href="<s:url action="contact"/>">Contact</a></li>
</ul>
```

Exemple d'arborescence



~/src/java/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="default" extends="struts-default">
      <result-types>
         <result-type name="tiles"
                class="org.apache.struts2.views.tiles.TilesResult" />
      </result-types>
      <action name="accueil" method="accueil"
                class="action.TilesAction">
          <result name="accueil" type="tiles">accueil</result>
      </action>
      <action name="contact" method="contact"
                class="action.TilesAction">
          <result name="contact" type="tiles">contact</result>
      </action>
      <action name="about" method="about"
                class="action.TilesAction">
          <result name="about" type="tiles">about</result>
      </action>
    </package>
</struts>
```

~/src/java/action/TilesAction.java

```
package action;
import com.opensymphony.xwork2.ActionSupport;

public class TilesAction extends ActionSupport {

    public String accueil() {
        return "accueil";
    }

    public String about() {
        return "about";
    }

    public String contact() {
        return "contact";
    }
}
```