[Le langage Java / Interfaces]

Les interfaces de marquage

- Elles servent simplement à indiquer une caractéristique de la classe
- Exemple : java.io.Serializable

[Le langage Java / Interfaces]

Les modificateurs

- public
  - Étend la portée au delà du paquetage
- abstract
  - Implicite ; peut être omis
- strictfp

[Le langage Java]

Les exceptions

- gestion aisée des cas d'erreurs
- pas d'alourdissement du code

[Le langage Java / Les exceptions]

Syntaxe en déclenchement


```
<déclaration de méthode> throws <exception>{
...
if (<condition>)
    throw new <exception>(message);
...
}
```

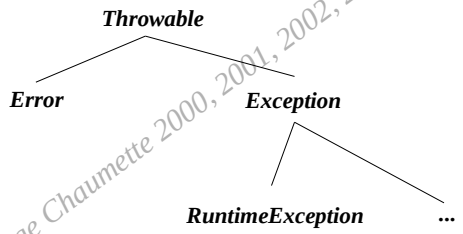
[Le langage Java / Les exceptions]

Syntaxe en traitement

```
try {
        ..
}
catch (type name){
        ...
}
catch(type name){
        ...
}
finally {
        ...
}
```


[Le langage Java / Les exceptions]

Les types d'exceptions



[Le langage Java / Les exceptions]

Exceptions contrôlées ou non

- Exceptions contrôlées
  - Elles doivent absolument être prises en compte dans l'application
  - Cela est vérifié par le compilateur
- Exceptions non contrôlées
  - Ne nécessitent pas obligatoirement d'être prises en compte par l'application
  - Elle étendent Error ou RuntimeException

[Le langage Java / Les exceptions]

Exemple 1

Melangeur.java

```
public class Melangeur{
        ...
        void melanger( ){
            for (int i=0; i<v1.size(); i++){
                try{
                  System.out.println(v1.elementAt(i) + ```` +  v2.elementAt(i));
                }catch (java.lang.ArrayIndexOutOfBoundsException ex){
                  System.out.println(``v2 est trop petit'');
                  return;
                }
            }
            System.out.println();
        }
...
```

[Le langage Java / Les exceptions]

Exemple 2

Melangeur2.java

```
public class Melangeur2{
        ...
        void melanger( ) throws VecteursIncompatiblesException{
            for (int i=0; i<v1.size(); i++){
              try{
                System.out.println(v1.elementAt(i) + `````` + v2.elementAt(i));
               }catch (java.lang.ArrayIndexOutOfBoundsException ex){
                  throw new VecteursIncompatiblesException(
                                                    "vecteur incompatibles");
               }
            }
            System.out.println();
        }
    ...
```


[Le langage Java / Les exceptions]

Exemple 2 (suite)

TestMelangeur2.java

```
...
        Vector v1=new Vector();
        v1.addElement("a"); v1.addElement("c");
        Vector v2=new Vector();
        v2.addElement("b");
        Melangeur2 melangeur=new Melangeur2(v1, v2);
        try{
            melangeur.melanger();
        } catch(VecteursIncompatiblesException ex){
            System.out.println("Melange impossible");
        }
    ...
```


Le ramasse-miettes

- Principe
- Finalisation
- Le problème de la résurrection
- Interaction avec le ramasse-miettes
- Les objets référence

[Le langage Java / Le ramasse-miettes]

Principe

- Allocation d'un objet
  - new
- Dé-allocation
  - Pas de delete
  - C'est le ramasse-miettes qui s'en charge
  - Un objet qui n'est plus référencé devient du *garbage* (rebut) ➔ *garbage collection*

Finalisation

- On peut implémenter

```
protected void finalize() throws Throwable;
```

- Appelée :
  - Une seule fois
  - Dans un délai non connu (voir jamais)
  - Pas de garantie / thread appelante
  - Attention aux objets qui ne sont plus référencés

Finalisation

- Exemple

```
protected void finalize(){

// finalisation de cette partie de l'objet

…

// finalisation de la partie de l'objet que l'on ne maîtrise pas
super.finalize();
}
```


Le problème de la résurrection

- Ressusciter un objet c'est le rendre à nouveau accessible lors de sa finalisation
- Problème : finalize ne sera plus re-invoquée
- Solution : créer un clone de l'objet

Interaction avec le ramasse-miettes

- Runtime.getRuntime() retourne une référence permettant d'invoquer les méthodes suivantes :

```
public void gc();
public void runFinalization();
public long freeMemory();
public long totalMemory();
```

- On peut aussi invoquer :

```
System.gc();
```

ou

```
System.runFinalization();
```

Le paquetage IO

- Présentation
- Les 2 hiérarchies de IO
- Les flots de caractères
- Les flots d'octets

Présentation

- java.lang.io regroupe toutes les classes permettant d'effectuer des entrées sorties
- Du plus bas niveau
- Au plus haut niveau

Les 2 hiérarchies de IO

- 2 hiérarchies
  - Flots de type caractères (*character stream*)
  - Flots de type octet (*byte stream*)

Les flots de caractères

Reader
- BufferedReader
  - LineNumberReader
- *CharArrayReader*
- InputStreamReader
  - *FileReader*
- FilterReader
  - PushbackReader
- *PipedReader*
- *StringReader*

[Le langage Java / Le paquetage IO]

Les flots de caractères

Writter
- BufferedWritter
- *CharArrayWriter*
- OutputStreamWriter
  - *FileWriter*
- FilterWriter
- *PipedWriter*
- *StringWriter*
- FilterWriter

[Le langage Java / Le paquetage IO]

Les flots d'octets

InputStream
- *FileInputStream*
- *PipedInputStream*
- FilterInputStream
  - LineNumberInputStream
  - DataInputStream
  - BufferedInputStream
  - PushbackInputStream
- *ByteArrayInputStream*
- SequenceInputStream
- *StringBufferInputStream*
- ObjectInputStream

[Le langage Java / Le paquetage IO]

Les flots d'octets

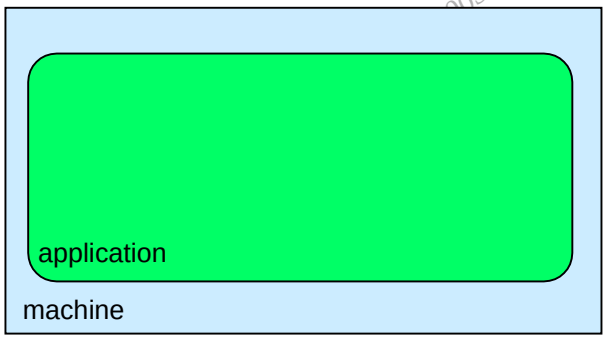
OutputStream
- *FileOutputStream*
- *PipedOutputStream*
- FilterOutputStream
  - DataOutputStream
  - BufferedOutputStream
  - PrintStream
- *ByteArrayoutputStream*
- ObjectOutputStream

Les *threads*

- Principe
- Opérations sur les *threads*
- Création d'une *thread*
- Actions sur les *threads*
- Priorités et ordonnancement
- Synchronisation
- Les groupes de *threads*

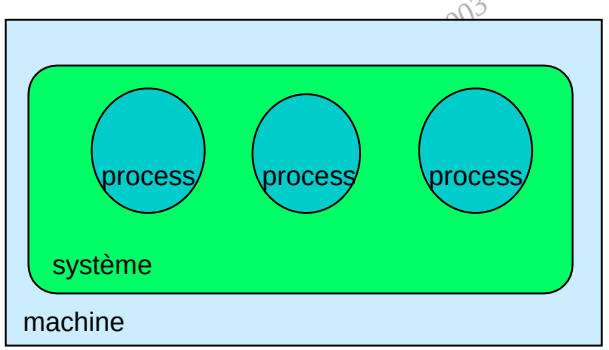
[Les *threads* / Principe]

La monoprogrammation



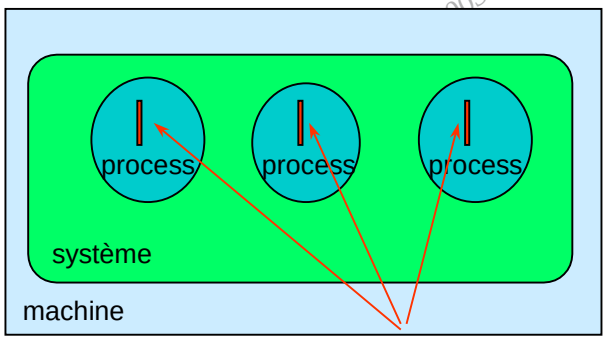
[Les *threads* / Principe]

La multiprogrammation ou multiprocessus



[Les *threads* / Principe]

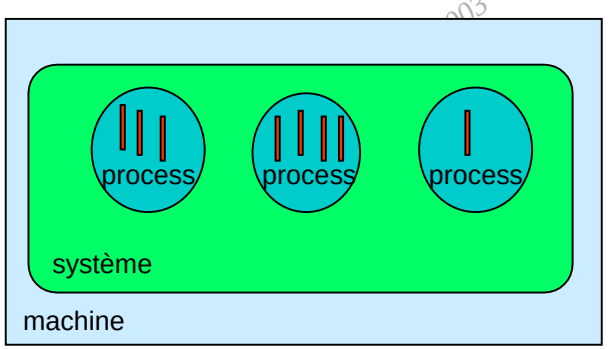
La multiprogrammation ou multiprocessus



1 *thread* / processus

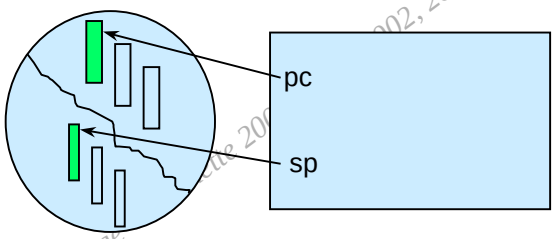
[Les *threads* / Principe]

Le multi*thread*



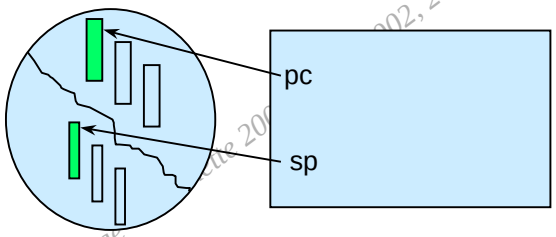
[Les *threads* / Principe]

Changement de *thread*



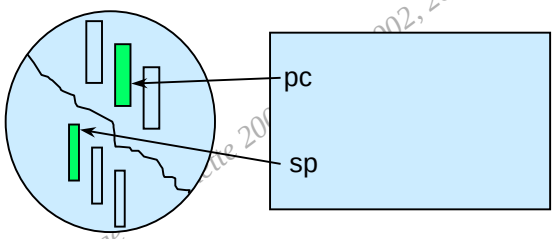
[Les *threads* / Principe]

Changement de *thread*



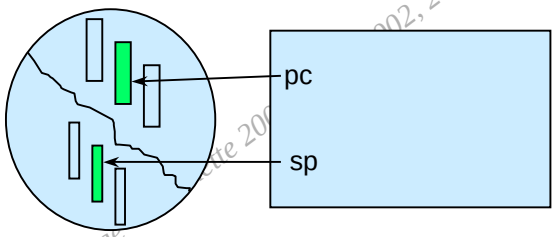
[Les *threads* / Principe]

Changement de *thread*



[Les *threads* / Principe]

Changement de *thread*



[Les *threads*]

Opérations sur les *threads*

- ➔ création
- ➔ action (arrêt, destruction, ...)
- ➔ ordonnancement, priorités
- ➔ synchronisation

[Les *threads*]

Création d'une *thread*

2 méthodes

- ➔sous classer *java.lang.Thread*
- ➔implementer l'interface *java.lang.Runnable*

[Les *threads* / Création d 'une *thread*]

Méthode 1

Main.java

```
class MaThread extends Thread{

    private String monNom;

    public MaThread(String nom){

        monNom=nom;
    }

    public void run(){

        for (int compteur=0; compteur < 10 ; compteur++)
            System.out.println(monNom + "[" + compteur + "]");
    }
}
```


[Les *threads* / Création d 'une *thread*]

Méthode 1 (suite)

Main.java (suite)

```
public class Main{

    static public main(String args[]){

        new MaThread("Premiere thread").start();
        new MaThread("Seconde thread").start();
    }
}
```

[Les *threads* / Création d 'une *thread*]

Méthode 2

Main.java

```
class MonRunnable implements Runnable{

    private String monNom;

    public MonRunnable(String nom){

        monNom=nom;
    }

    public void run(){

        for (int compteur=0; compteur < 10 ; compteur++)
            System.out.println(monNom + "[" + compteur + "]");
    }
}
```


[Les *threads* / Création d 'une *thread*]

Méthode 2 (suite)

Main.java (suite)

```
public class Main{

    static public void main(String args[]){

        new Thread(new MonRunnable("Premier runnable")).start();
        new Thread(new MonRunnable("Second runnable")).start();
    }
}
```


[Les *threads*]

Actions sur les *threads*

- *start / stop*
- *suspend / resume*
- *sleep*
- *yield*
- *join*
- *destroy*

[Les *threads* / Actions sur les *threads*]

Méthodes à problèmes

- *start / ~~stop~~*
- *~~suspend~~ / ~~resume~~* incohérences
- *sleep*
- *yield*
- *join*
- *destroy*

Threads et interruptions

Les méthodes qui posent des problèmes ont été abandonnées au profit de :

- *interrupt*
- *interrupted* / *isInterrupted*

Threads et interruptions : exemple

Main.java

```
class MaThread extends Thread{

   private String monNom;

   public MaThread(String nom){
         monNom=nom;
   }

   public void run(){
         for (int compteur=0; compteur < 100 ; compteur++){
            if (isInterrupted()) return;
            System.out.println(monNom + "[" + compteur + "]");
         }
   }
}
```


Threads et interruptions : exemple

Main.java (suite)

```
public class Main{

   static public void main(String args[]){
         Thread t=new MaThread("Premiere thread");
         t.setPriority(Thread.MIN_PRIORITY);
         t.start();
         try{
            Thread.currentThread().sleep(30);
         } catch (java.lang.InterruptedException e){
            System.out.println(e.getMessage());
         }
         t.interrupt();
   }
   ...
```


La methode *yield*

Main.java

```
class MaThread extends Thread{
...
      public void run(){

         for (int compteur=0; compteur < 10 ; compteur++){
            System.out.println(monNom + "[" + compteur + "]");
            yield();
         }
}
```

[Les *threads*]

Priorités et ordonnancement

- Ordonnancement
  - par rapport aux priorités
  - *time-slicing* non spécifié
- Priorités
  - *MIN_PRIORITY* / *MAX_PRIORITY*

[Les *threads* / Priorités et ordonnancement]

Exemple

Main.java

```
class MaThread extends Thread{
...
        public void run(){

                for (int compteur=0; compteur < 10 ; compteur++){
                   System.out.println(monNom + "[" + compteur + "]");
                   if (compteur==5)
                           this.setPriority(Thread.MIN_PRIORITY);
                   }
}
```


[Les *threads*]

Synchronisation

- sections critiques
  - *synchronized*
- verrous
  - *wait* / *notify*

[Les *threads* / synchronisation]

Portions de code synchronisées

```
...
synchronized(<objet>)
        <instruction>;
...
```

- on verrouille <*objet*>
- on exécute <*instruction*>
- on déverrouille <*objet*>

[Les *threads* / synchronisation]

Méthodes synchronisées

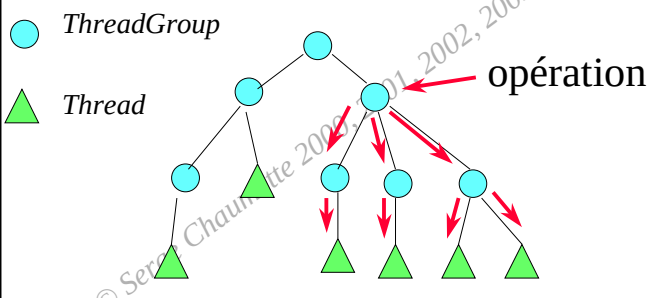
```
public class Demo{
...
    synchronized public int <methode>(){
        ...
    }
...
}
```

quand on invoque <*objet*>.<*methode*>()

- on verrouille <*objet*>
- on exécute <*methode*>
- on déverrouille <*objet*>

[Les *threads*]

Les groupes de *threads*



La communication par *sockets*

- Rappels
- Exemple : un client *Finger*
- Abstraction et exemple
- Autres fonctionnalités

[La communication par *sockets* / Rappels]

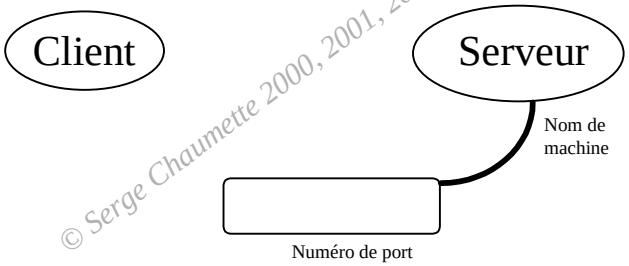
Bas niveau (*POSIX*)

côté client	côté serveur
socket	*socket*
bind	*listen*
connect	*accept*
→ descripteur (*read*/*write*)	→ descripteur (*read*/*write*)

Rappels sur les *sockets*

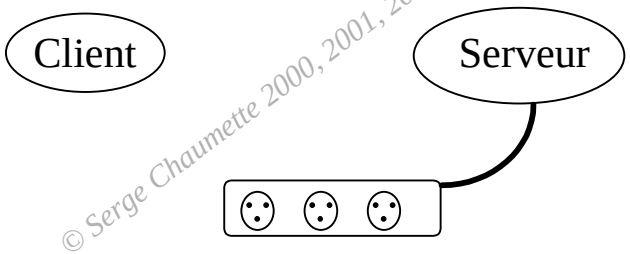
Rappels sur les *sockets*

serveur : *socket*



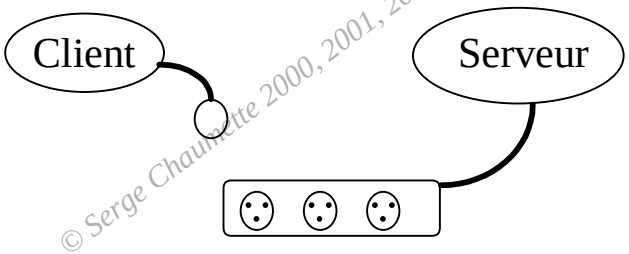
Rappels sur les *sockets*

serveur : *listen*(3)



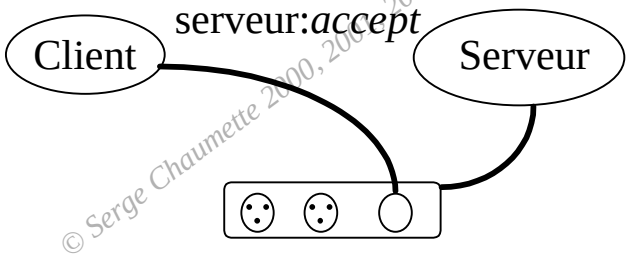
Rappels sur les *sockets*

client : *socket*



Rappels sur les *sockets*

client : *bind*, *connect*
serveur:*accept*



Bas niveau (*POSIX*)

côté client	côté serveur
socket	*socket*
bind	*listen*
connect	*accept*
→ descripteur (*read*/*write*)	→ descripteur (*read*/*write*)

Exemple : un client *Finger*

Finger.java

```
...
    public static void main(String args[])
            throws IOException, UnknownHostException {

        Socket socket = new Socket(args[1], 79);

        DataOutputStream sortie=new DataOutputStream(socket.getOutputStream());
        DataInputStream entree=new DataInputStream(socket.getInputStream());

        sortie.writeBytes(args[0] + "\n");
        String ligne;
        while ((ligne=entree.readLine())!=null) System.out.println(ligne);
    }
...
```


Abstraction côté serveur

Mise en place d'un point de communication

```
ServerSocket serverSocket= new ServerSocket(<numero de port TCP>);
```

Attente d'une demande de connexion

```
Socket socket=serverSocket.accept();
```

Récupération des flots (communication bidirectionnelle)

```
InputStream entree=socket.getInputStream();
OutputStream sortie=socket.getOutputStream();
```

Exemple : le serveur

Serveur.java

```
...
        ServerSocket serverSocket= new ServerSocket(numeroDePort);

        while (true){
                Socket socket=serverSocket.accept();

                InputStream entree=socket.getInputStream();
                OutputStream sortie=socket.getOutputStream();

                int entier1=entree.read();
                int entier2=entree.read();
                sortie.write(entier1+entier2);
        }
...
```


Abstraction côté client

Connexion au serveur

Socket socket= new **Socket**(*<nom ip de machine>*, *<numero de port TCP>*);

Recuperation des flots (communication bidirectionnelle)

InputStream entree=socket.**getInputStream**();
OutputStreamentree=socket.**getOutputStream**();

Exemple : le client

Client.java

```
...
        Socket socket= new Socket(nomDuServeur, numeroDePort);

        InputStream entrée=socket.getInputStream();
        OutputStream sortie=socket.getOutputStream();

        sortie.write(Integer.parseInt(args[2]));
        sortie.write(Integer.parseInt(args[3]));
        System.out.println(entree.read());
...
```


Exécution

Côté serveur

```
$ java Serveur 3000 &
$
```

Côté client

```
$ java Client jupiter.labri.u-bordeaux.fr 3000 41 36
77
$
```

[La communication par *sockets*]

Autres fonctionnalités

- *Sockets UDP*
- *Multicast*

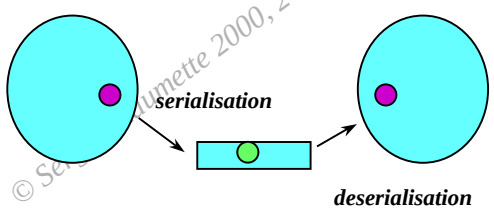
Serialisation

- Principe
- Utilisations possibles
- Exemple
- Ce qui est *serialise*
- Mise en œuvre
- Gestion par la classe
- Contrôles supplémentaires

[*Serialisation*]

Principe

La *serialisation* permet d'écrire/lire l'état d'un objet dans un flot.



[*Serialisation*]

Utilisations possibles

- objets persistants
- points de reprise
- transmission d'objets entre applications
  - locales
  - distantes

[*Serialisation* / Exemple]

La *serialisation*

Sauver.java

```
...
Evenement evenement=new Evenement();
evenement.debut();
for (int i=0; i<1000; i++) java.lang.System.gc();
evenement.fin();

FileOutputStream fos=new FileOutputStream(``evenement.ser");

ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(evenement);
oos.flush();
oos.close();
fos.close();
...
```


[*Serialisation* /Exemple]

La *deserialisation*

Restaurer.java

```
...
FileInputStream fis=new FileInputStream(``evenement.ser");

ObjectInputStream ois=new ObjectInputStream(fis);
 try{
        Evenement evenement = (Evenement)ois.readObject();
        System.out.println(evenement);
} catch (java.lang.ClassNotFoundException ex){};
 ois.close();
 fis.close();
...
```

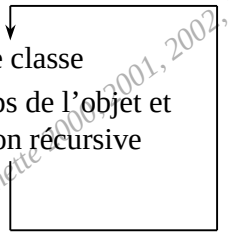

[*Serialisation* / Exemple]

Exécution

```
$ java Sauver
$ cat evenement.ser
¬ísr    EvenementT,_cv÷L      dateDebuttLjava/util/Date;
LdateFinq~xpsrjava.util.DatehjKYtxpÓ1ñxsq~Ó1  Fx
$ java Restaurer
Tue Sep 29 12:30:33 GMT+01:00 1998 <----->
 Tue Sep 29 12:30:39 GMT+01:00 1998
$
```


[*Serialisation*]

Ce qui est *serialisé*



- le nom de classe
- les champs de l'objet et ce de façon récursive

[*Serialisation* / Ce qui est *serialisé*]

Les champs *serialisés*

- par défaut :
  - les champs non *static*
  - les champs non *transient*
- paramétrable via le champs *Fields*

[*Serialisation*]

Mise en œuvre

- la classe à *serialiser* doit :
  - implementer l'interface *java.io.Serializable*
  - [éventuellement] déclarer ses champs *serialisables*
- la classe à *serialiser* peut :
  - gérer sa *serialisation*
  - implementer des mécanismes de contrôle/sécurité

[*Serialisation*]

Gestion par la classe

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException;

private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```


[*Serialisation* / Gestion par la classe]

Exemple

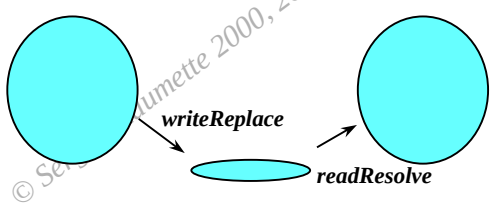
Evenement.java

```
...
public class Evenement implements Serializable{
...
        public void readObject(ObjectInputStream ois){
            try{
                ois.defaultReadObject();
             } catch (java.lang.ClassNotFoundException ex){
                    throw new InvalidObjectException(ex.getMessage());
            };
            if (dateDebut.compareTo(dateFin)>=0)
                    throw new InvalidObjectException("Dates invalides");
        }
...
}
```

[*Serialisation* / Contrôles supplémentaires]

Resolvable et *Replacable*

Il existe deux méthodes supplémentaires, ***writeReplace*** et ***readResolve***. Elles permettent a un objet d'indiquer quel autre objet sauver/restaurer en ses lieux et places.



[*Serialisation* / Contrôles supplémentaires]

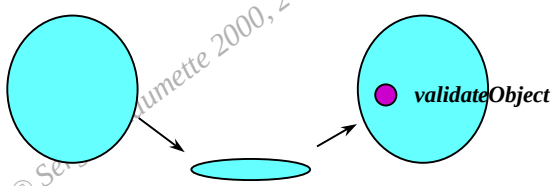
Resolvable et *Replacable*

```
<modificateurs> Object writeReplace()
                throws ObjectStreamException;

<modificateurs> Object readResolve()
                throws ObjectStreamException;
```


[*Serialisation* / Contrôles supplémentaires]

Interface *ObjectInputValidation*

L'interface *ObjectInputValidation* permet à un objet d'être appelé quand il a été restauré.



[*Serialisation* / Contrôles supplémentaires]

Interface *ObjectInputValidation*

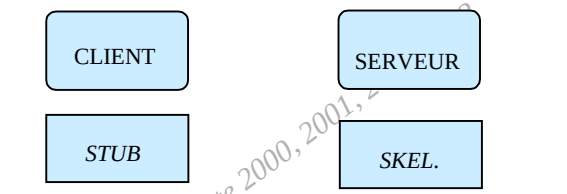
```
public interface ObjectInputValidation{

        void validateObject() throws InvalidObjectException;
}
```

Invocation de méthodes distantes

- Principe
- Exemple
  - Description de l'interface
  - Description de l' implémentation
  - Génération du *stub*/*skeleton*
  - Réalisation d'un client
  - Réalisation du serveur
  - Exécution
- Sécurité

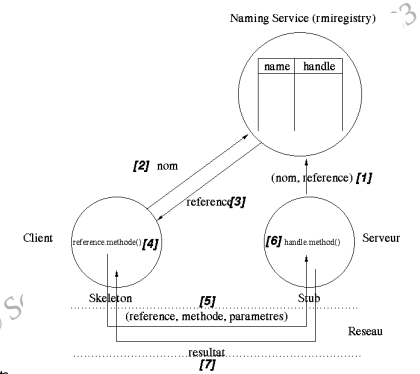
[Invocation de méthodes distantes]

Principe



[Invocation de méthodes distantes]

Principe (suite)



[Invocation de méthodes distantes / Exemple]

Description de l' interface

Calculette.java

```
public interface Calculette extends java.rmi.Remote{

   int ajouter(int valeur1, int valeur2) throws java.rmi.RemoteException;
}
```

[Invocation de méthodes distantes / Exemple]

Description de l'implémentation

CalculetteImpl.java

```
import java.rmi.server.UnicastRemoteObject;

public class CalculetteImpl extends UnicastRemoteObject
        implements Calculette{

    public int ajouter(int valeur1, int valeur2){
        return valeur1+valeur2;
    }
}
```


[Invocation de méthodes distantes / Exemple]

Generation du *stub/skeleton*

```
$ javac  Calculette.java
$ javac  CalculetteImpl.java
$
$ ls
CalculetteImpl.class    Calculette.class
$
$ rmic CalculetteImpl
$
$ ls
CalculetteImpl.class      CalculetteImpl_Skel.class
CalculetteImpl_Stub.class       Calculette.class
$
```


[Invocation de méthodes distantes / Exemple]

Réalisation d'un client

TestCalculette.java

```
...
        Calculette calculette =
                (Calculette) java.rmi.Naming.lookup
                    ("//kediri.jodo.labri.u-bordeaux.fr/Calculette");
        int resultat=calculette.ajouter(20, 30);
        System.out.println(resultat);
...
```


[Invocation de méthodes distantes / Exemple]

Réalisation du serveur

Serveur.java

```
...
        System.setSecurityManager(new java.rmi.RMISecurityManager());
...
        CalculetteImpl calculetteImpl = new CalculetteImpl();
        java.rmi.Naming.bind( ``Calculette", calculetteImpl);
...
```

[Invocation de méthodes distantes / Exemple]

Exécution

Côté serveur

```
$ rmiregistry &
$ java -Djava.security.policy=policy.v1 Serveur &
$
```

Côté client

```
$ java TestCalculette
50
$
```


[Invocation de méthodes distantes]

Sécurité : problème d'accès

Côté serveur

```
$ rmiregistry &
$
java Serveur
access denied (java.net.SocketPermission 127.0.0.1:1099 connect,resolve)
^C
$
```


[Invocation de méthodes distantes / Sécurité]

Fichier de configuration

policy.v1

```
grant {
        // On autorise tout
        permission java.security.AllPermission;
};
```


[Invocation de méthodes distantes / Sécurité]

Exécution

Côté serveur

```
$ rmiregistry &
$ java -Djava.security.policy=policy.v1 Serveur &
$
```

Côté client

```
$ java TestCalculette
50
$
```

Sécurité : problème d'accès

```
$ java Serveur
 access denied (java.net.SocketPermission 127.0.0.1:1099 connect,resolve)
^C
$
```

http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html

Fichier de configuration

policy.v2

```
grant {
      permission
        java.net.SocketPermission
        "kediri.jodo.labri.u-bordeaux.fr:1099", "connect,resolve";
};
```


Exécution

Côté serveur

```
$ rmiregistry &
$ java -Djava.security.policy=policy.v2 Serveur &
$
```

Côté client

```
$ java TestCalculette
50
$
```


Codes (agents) mobiles

- Principe
- Un agent
- Le serveur
- Exemple d'agent

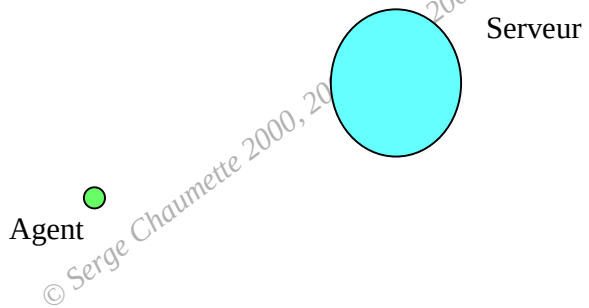
[Codes (agents) mobiles]

Principe

- ➔migration
  - ♦ *serialisation*
  - ♦ invocation de méthodes distantes
- ➔communication
  - ♦ invocation de méthodes distantes
  - ♦ communication par *sockets*
- ➔activité
  - ♦ *threads*

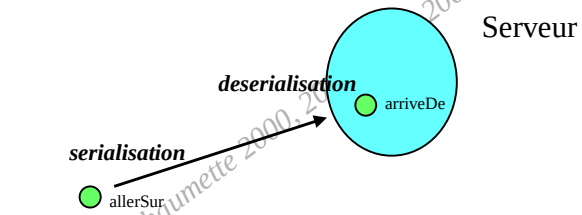
[Codes (agents) mobiles]

Principe



[Codes (agents) mobiles]

Principe



[Codes (agents) mobiles]

Un agent

- ➔A une méthode *allerSur*
  - ♦ elle le *serialise* et l'envoie à un serveur
- ➔ A une méthode *arriveDe*
  - ♦ elle est invoquée par le serveur à son arrivée sur une machine