[Codes (agents) mobiles / Un agent]

Code Java

Agent.java

```
…
public abstract class Agent implements Serialisable{

        public final void allerSur(String adresseURL){

        // On serialise l'agent (i.e. this) vers la machine indiquee
        …
        s.writeObject(this);
        ...
        }

        public abstract void arriveDe(String machineSource);
}
```


[Codes (agents) mobiles]

Le serveur

- Attend une connexion d'un agent
- Lit un agent sur la *socket* (*deserialisation*)
- Invoque sa méthode *arriveDe*

[Codes (agents) mobiles / Le serveur]

Code Java

Serveur.java

```
class ServeurAgents{
…
        public void traiterRequete() throws java.io.IOException,
                                        java.lang.ClassNotFoundException{

          Socket socket = serveurSocket.accept();
          // lecture de l'  agent sur la socket
           ...
          Agent agent = (Agent) s.readObject();
           ...
           // activation de l'  agent
          agent.arriveDe(machineSource);
        }
}
```


[Codes (agents) mobiles]

Exemple d'agent

MonAgent.java

```
import java.io.Serializable;

public class MonAgent extends Agent implements Serializable{

   String machines[]={"http://kediri:5000", "http://cafebabe:5000"};

   private int i=0;

   public MonAgent(){

      allerSur("http://kediri:5000");
   }

   ...
```

[Codes (agents) mobiles]

Exemple d'agent (suite)

MonAgent.java (suite)

```
...
   public void arriveDe(String machine){

      System.out.println("J'  arrive de la machine " + machine);
      i=1-i;
      allerSur(machines[i]);
   }

   static public void main(String args[]){

      Agent agent=new MonAgent();
      ...
   }
```


[Codes (agents) mobiles]

Exécution

Sur *cafebabe*

```
$ java Serveur 5000 &
$J'  arrive de la machine kediri
J'  arrive de la machine kediri
...
```

Sur *kediri*

```
$ java Serveur 5000 &
$ java MonAgent &
$J'  arrive de la machine cafebabe
J'  arrive de la machine cafebabe
...
```

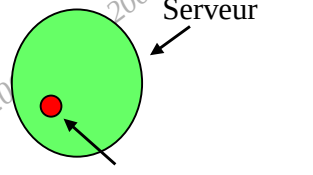

Jini

- Principe
- *Discovery* / *join* / *lookup*
- Phase de *discovery*
- Phase de *join*
- Phase de *lookup*
- Phase d'utilisation

[*Jini*]

Principe

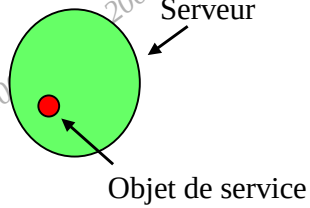
- Client/serveur
- Service
  - Serveur
  - Objet de service



[*Jini*]

Principe

- Client/serveur
- Service
  - Serveur
  - Objet de service
- Client
  - Dialogue avec/via l'objet de service

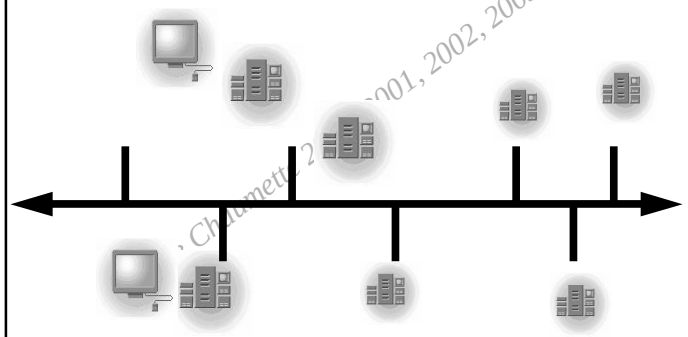


[*Jini*]

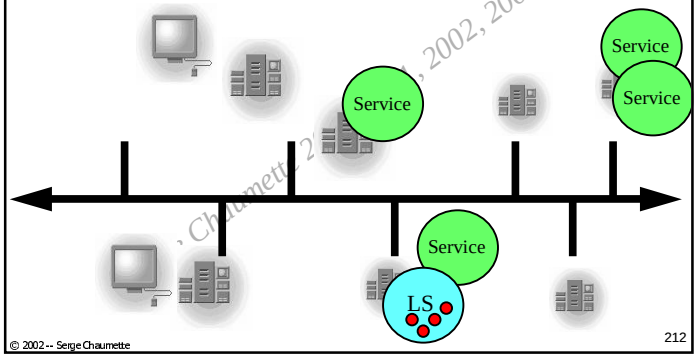
Discovery / join / lookup

- *Discovery*
  - Recherche d'un service de *lookup*
- *Join*
  - Le service rejoint la communauté Jini
  - Enregistrement d'un objet de service
- *Lookup*
  - Recherche d'un service
  - Récupération de l'objet de service

[*Jini*]

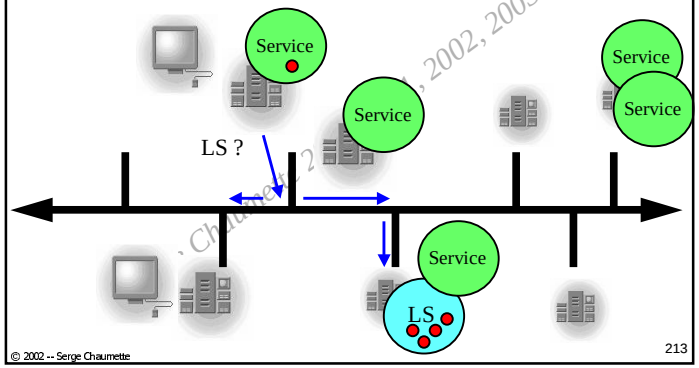


[*Jini*]



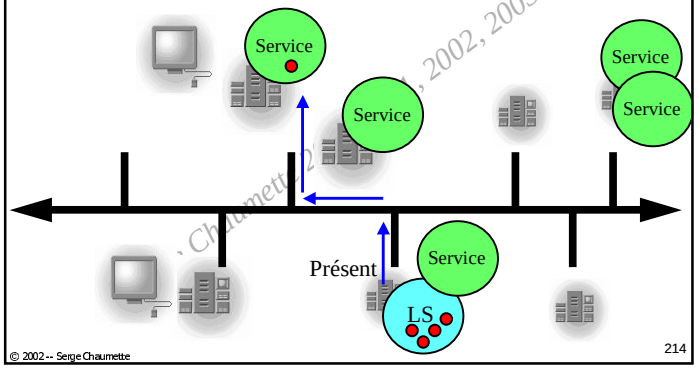
[*Jini*]

Phase de *discovery*



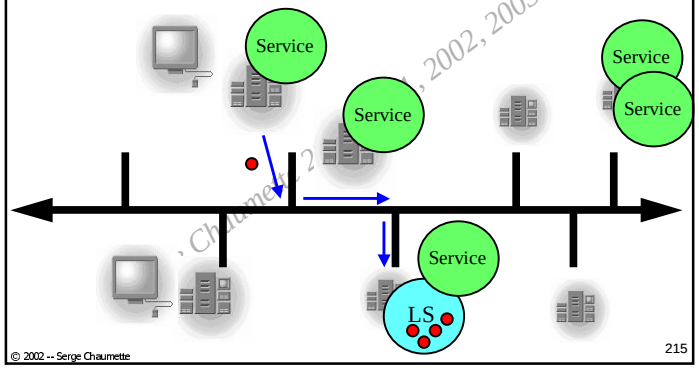
[*Jini*]

Phase de *discovery*



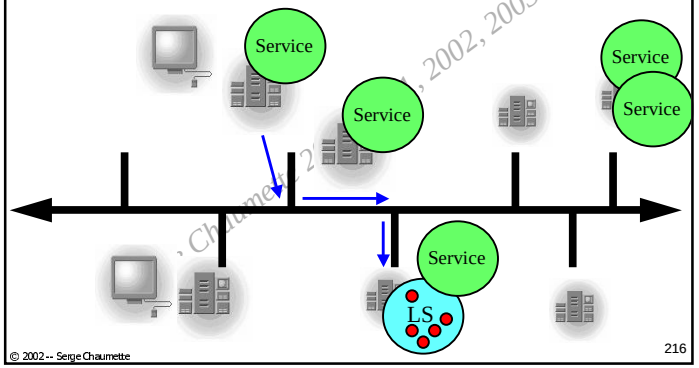
[*Jini*]

Phase de *join*



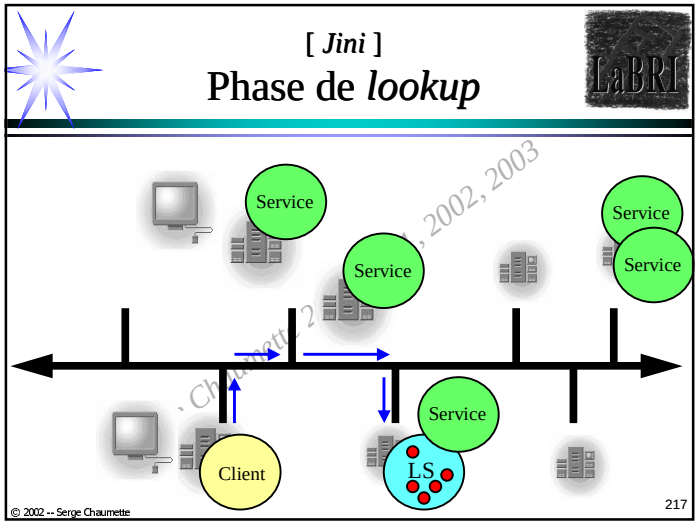
[*Jini*]

Phase de *join*



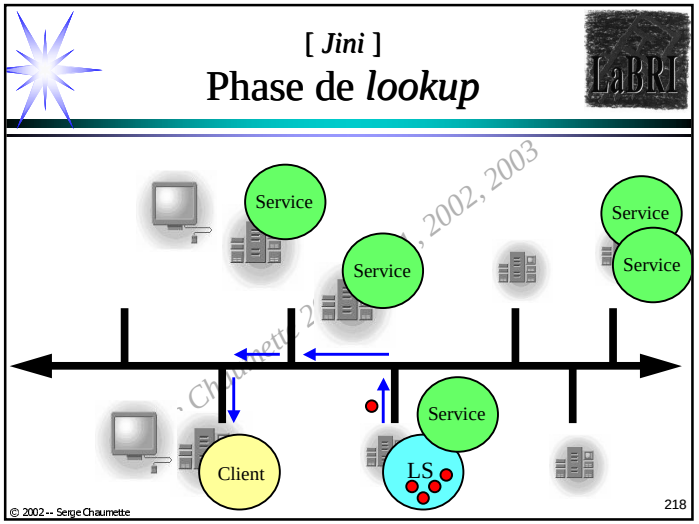
[*Jini*]

Phase de *lookup*



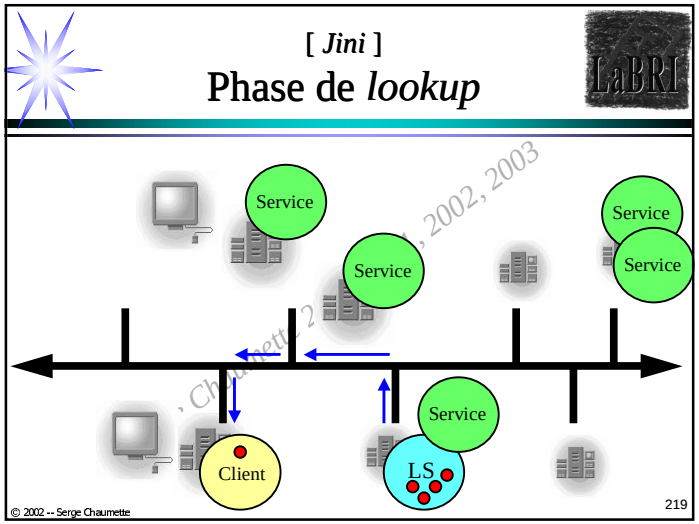
[*Jini*]

Phase de *lookup*



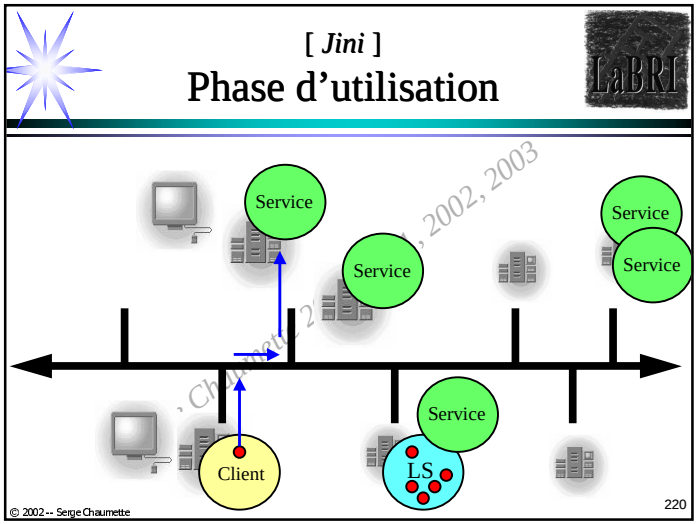
[*Jini*]

Phase de *lookup*



[*Jini*]

Phase d'utilisation

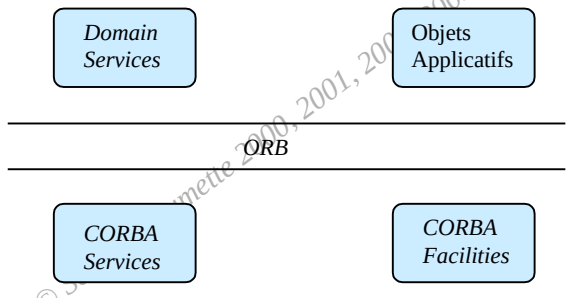


CORBA

- *Object Management Architecture*
- Principe
- Exemple
- Principe avec *BOA*
- Le service de nommage
- Implémentations
- Communication inter-*ORB*s

[*CORBA*]

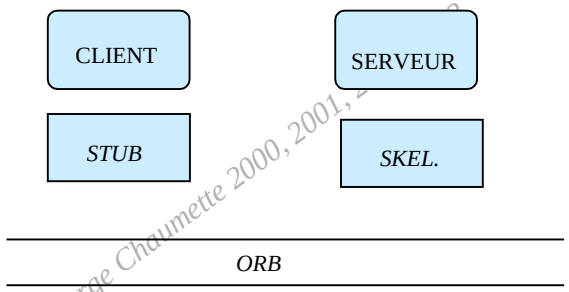
Object Management Architecture



Common Object Request Broker Architecture

[*CORBA*]

Principe



[*CORBA*]

Exemple

Une calculette capable d'ajouter deux valeurs.

- décrire le service
- écrire son implémentation
- écrire un serveur

- écrire un client

[*CORBA* / Exemple]
Description du service

On utilise un langage appelé *IDL* (*Interface Definition Language*)

- ➔ indépendant du langage d'implémentation
- ➔ indépendant du système
- ➔ indépendant de l' implémentation de *CORBA*

On ne décrit que l'interface

[*CORBA* / Exemple]
Description IDL du service

Calculette.idl

```
module labri {
    module corba {
        interface Calculette {
            long ajouter(in long valeur1, in long valeur2);
        }
    }
}
```


[*CORBA* / Exemple]
Traduction de l'*IDL*

Calculette.idl

```
$ idltojava Calculette.idl
$
$ ls labri/corba
Calculette.java
CalculetteHelper.java
CalculetteHolder.java
_CalculetteImplBase.java
_CalculetteStub.java
$
```


[*CORBA* / Exemple]
Interface Java générée

labri/corba/Calculette.java

```
/*
 * File: ./labri/corba/Calculette.java
 * From: Calculette.idl
 * Date: Tue Sep 29 13:12:58 1998
 *   By: idltojava Java IDL 1.2 Nov 12 1997 12:23:47
 */

package labri.corba;
public interface Calculette
    extends org.omg.CORBA.Object {
    int ajouter(int valeur1, int valeur2)
;
}
```

[*CORBA* / Exemple]

Le serveur

Serveur.java

```
class CalculetteServant extends _CalculetteImplBase {

   public int ajouter(int valeur1, int valeur2){

      return valeur1 + valeur2;
   }
}
```


[*CORBA* / Exemple]

Le serveur (suite)

Serveur.java (suite)

```
public class Serveur {
…
        // cree et initialise l' ORB
        ORB orb = ORB.init(args, null);

        // cree le servant et le declare a l'ORB
        CalculetteServant calculetteRef = new CalculetteServant();
        orb.connect(calculetteRef);

        String ior=orb.object_to_string(calculetteRef);
        System.out.println(ior);

        // attend les invocations des clients
...
```


[*CORBA* / Exemple]

Le client

Client.java

```
public class Client {

   public static void main(String args[]){
   …
        // cree et initialise l' ORB
        ORB orb = ORB.init(args, null);

        org.omg.CORBA.Object obj=orb.string_to_object(args[0]);
        Calculette clientRef = CalculetteHelper.narrow(obj);

        // invocation et affichage du resultat
        int result = clientRef.ajouter(10, 23);
        System.out.println("Resultat  : " + result);
```


[CORBA / Exemple]

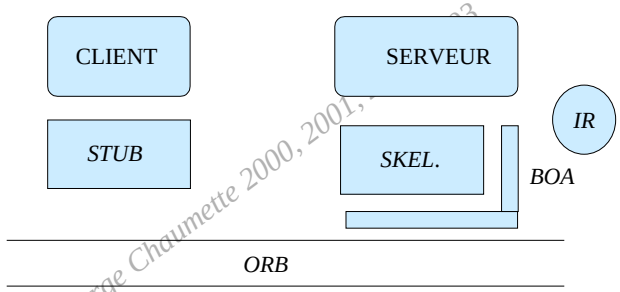
Exécution

```
$ java Serveur -ORBInitialPort 1050 &
IOR:000000000000001f49444c3a6c616272692f636f7262612f436
16c63756c657474653a312e3000000000000100000000000000300
001000000000076b6564697269000083bb000000000018afabcafe
000000021e944a0800000008000000000000000
$

$ java Client IOR:... -ORBInitialPort 1050
Resultat  : 33
$
```

[*CORBA*]

Principe avec *BOA*



[*CORBA*]

Principe avec *BOA* (suite)

Le *BOA*, *Basic Object Adapter*, assure l'interface entre le système d'exploitation et les services. Il décide quand et comment activer les implémentations.

Les informations nécessaires sont enregistrées dans l'*IR*, *Implementation Repository*, par les serveurs.

[*CORBA*]

Le service de nommage

Le service de nommage permet d' associer un ou plusieurs noms logiques à une référence physique.

Le service de nommage est lui-même un service *CORBA*.

[*CORBA* / Le service de nommage]

Structure arborescente

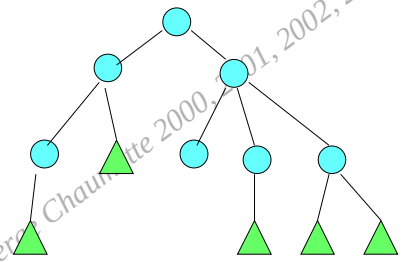


Schéma d'utilisation

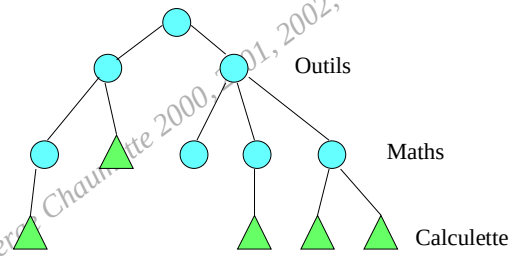
côté client

```
// Initialisation de l' ORB
ORB orb = ORB.init(...);
// Acces au service de nommage
orb.resolve_initial_references
// Resolution
nc.resolve(...)
```

côté serveur

```
// Initialisation de l' ORB
ORB orb = ORB.init(...);
// Acces au service de nommage
orb.resolve_initial_references
// Liaison
nc.bind(...)
```


Structure arborescente



Côté serveur

Serveur.java

```
…
Orb orb=ORB.init(args, null);

CalculetteServant calculetteRef=new CalculetteServant();
orb.connect(calculetteRef);

org.omg.CORBA.Object obj=
        obj=orb.resolve_initial_references("NameService");
NamingContext nc=NamingContextHelper.narrow(obj);

NameComponent c1 = new NameComponent("Outils", "");
NameComponent chemin1[]={c1};
nc=NamingContextHelper.narrow(nc.bind_new_context(chemin1));
…
```


Côté serveur (suite)

Serveur.java (suite)

```
…
NameComponent c2 = new NameComponent("Maths", "");
NameComponent chemin2[]={c2};
nc=NamingContextHelper.narrow(nc.bind_new_context(chemin2));

NameComponent c1 = new NameComponent("Calculette", "");
NameComponent chemin[]={c3};
nc.rebind(chemin3, calculetteRef);
…
```

Côté client

Client.java

```
…
Orb orb=ORB.init(args, null);

org.omg.CORBA.Object obj=
        obj=orb.resolve_initial_references("NameService");
NamingContext nc=NamingContextHelper.narrow(obj);

NameComponent c1 = new NameComponent("Outils", "");
NameComponent c2 = new NameComponent("Maths", "");
NameComponent c3 = new NameComponent("Calculette", "");
NameComponent chemin[]={c1, c2, c3};
obj=nc.resolve(chemin);

Calculette clientRef= CalculetteHelper.narrow(obj);
...
```


Lancement du service

Java IDL

```
$ tnameserv -ORBIntialPort 1050 &
```


Lancement du service (suite)

VisiBroker

```
$ vbj -DORBservices=CosNaming \
      com.visigenic.vbroker.services.CosNaming.ExtFactory \
      Outils  log &

$ vbj -DORBservices=CosNaming \
      -DSVCnameroot Outils \
      com.visigenic.vbroker.services.CosNaming.ExtFactory \
      Maths  log &
$
```


Lancement du service (suite)

VisiBroker

```
$ vbj -DORBservices=CosNaming \
      -DSVCnameroot Outils:Maths \
      com.visigenic.vbroker.services.CosNaming.ExtFactory \
      Simples  log &
$
```

[CORBA / Le service de nommage / Utilisation]

Lancement du client (suite)

VisiBroker

```
$ vbj -DORBservices=CosNaming \
      -DSVCnameroot Outils:Maths/Simples \
      Calculette log &
$
```


[CORBA / Le service de nommage]

Interfaces

- ➔ *NamingContext*
- ➔ *NameComponent*

Ces interfaces sont décrites en *IDL* : le service de nommage est un service *CORBA*

[*CORBA*]

Implémentations

- IONA : OrbixWeb
- Visigenix : VisiBroker
- Chorus : CoolORB
- Sun : Java IDL

http://www.omg.org/

[*CORBA* / Implémentations]

Java IDL

Implémentation de *Sun*

http://www.javasoft.com/products/idl/

C'est une implémentation minimale :

- *ORB*
- *idltojava*
- service de nommage
- *IIOP*

[*CORBA*]

Communication inter-*ORB*s

- *GIOP* (*General Inter-ORB Protocol*)
  - messages échanges
  - format des données
- *IIOP* (*Internet Inter-ORB Protocol*)
  - réalisation des messages *GIOP* sur *Internet*

Java IDL implemente *IIOP*.

Conclusion

Java RMI vs CORBA

- RMI, sérialisation
  - Java vers Java + IIOP
  - Appel par valeur ou par référence
  - Passage d'objets contenant du code
  - Verifications de sécurité
- CORBA
  - Interoperabilité
  - Passage de paramètres in, out, inout
  - Pas de passage d'objets contenant du code
  - Verifications de sécurité fonction de l'ORB

[Conclusion]

Java RMI vs CORBA

La réflexion

- Principe
- Quelques utilisations potentielles
- La classe Class
- La classe Field
- La classe Method

[La réflexion]

Principe

- Un objet se décrit vers l'extérieur
- Nouvelles classes
  - java.lang.Class
  - java.reflect.Method
  - Java.lang.reflect.Field
  - …
- Principe
  - Class.forName(<*nom de classe*>) retourne un objet de type Class qui décrit la classe indiquée

[La réflexion]

Quelques utilisations potentielles

- Invocation de méthodes à distance
- Gestion des communications asynchrones
  - Création de files d'attente des requêtes
  - Passer de la requête à l'invocation de méthode
- Écriture d'un interpréteur
- …

[La réflexion]

La classe Class

- Obtention d'information relatives au type de la classe
  - boolean isInterface();
  - boolean isPrimitive();
  - Pour les tableaux
    - boolean isArray();
    - Class getComponentType();

[La réflexion / La classe Class]

Exemple

Exemple1.java

```
public class Interface{

    static public void main(String args[])
            throws java.lang.ClassNotFoundException{

        Class c=Class.forName(args[0]);
        System.out.println(c.isInterface());
    }
}
```

[La réflexion]

La classe Class

- Accès aux informations relatives à la déclaration de la classe
  - package getPackage();
  - Class[] getInterfaces();
  - Class getSuperClass();
  - int getModifiers();

[La réflexion]

La classe Class

- Compatibilités
  - boolean isAssignableFrom(Object o);
- Autres
  - String getName();
  - Object newInstance();

[La réflexion / La classe Class]

Exemple

Creation.java

```
public class Creation{

   static public void main(String args[])
         throws java.lang.ClassNotFoundException,
              java.lang.IllegalAccessException,
              java.lang.InstantiationException{

         Class c=Class.forName(args[0]);
         Object o=c.newInstance();
         System.out.println(o.toString());
      }
   }
```


[La réflexion / La classe Class]

Obtention d'un objet de type Class

- méthode getClass de java.lang.Object
  - Class getClass();
- champ .class d'une classe
- méthode statique forName de Class
  - static Class.forName(String nom)
- Par réflexion, c'est à dire comme résultat d'une interrogation d'une classe

[La réflexion / La classe Class]

Cas particulier des types de base

- Class.forName n'existe naturellement pas pour les types de base
  - Integer.TYPE
  - int.class

[La réflexion / La classe Class]

Exemple

Exemple1.java

```
public class Exemple1{

  static public void main(String args[])
        throws java.lang.ClassNotFoundException{

        Object stack=new java.util.Stack();
        System.out.println(stack.getClass().getName());

        System.out.println(Class.forName("java.util.Vector").getName());

        System.out.println(int.class.getName());
        System.out.println(Integer.TYPE.getName());
    }
  }
```


[La réflexion / La classe Class]

Accès aux membres

- Membres publics déclarés ou hérités
  - Constructor[] getConstructors();
  - Method[] getMethods();
  - Field[] getFields();
- Tous les membres déclarés (non hérités)
  - Constructor[] getDeclaredConstructors();
  - Method[] getDeclaredMethods();
  - Field[] getDeclaredFields();

[La réflexion / La classe Class / Accès aux membres]

Exemple

Liste.java

```
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Liste{

  static public void main(String args[])
        throws java.lang.ClassNotFoundException{

        Class c=Class.forName(args[0]);
        Field[] fields=c.getFields();
        for (int i=0; i<fields.length; i++)
          System.out.println(fields[i].getName());

        Method[] methods=c.getMethods();
        for (int i=0; i<methods.length; i++)
          System.out.println(methods[i].getName());
    }
  }
```

[La réflexion]

La classe Field

- Décrit un champ de classe, d'instance ou d'une interface
- Description du champs
  - int getModifiers();
  - String getName();
  - Class getType();

[La réflexion / La classe Field]

Exemple

Exemple2.java

```
import java.lang.reflect.Field;

public class Exemple2{
   static public void main(String args[])
         throws java.lang.ClassNotFoundException,
              java.lang.NoSuchFieldException{

         Class c=Class.forName(args[0]);
         Field f=c.getField(args[1]);
         Class type=f.getType();
         System.out.println(type.getName());
   }
}
```


[La réflexion]

La classe Field

- Modification/lecture de la valeur
  - Object get(Object o);
  - void set(Object o, Object valeur);
  - <type> get<Type>(Object o);
  - void set<Type>(Object o, <type> valeur);

[La réflexion]

La classe Method

- Elle décrit et fournit un accès à une méthode d'une classe ou d'une interface
- Accès à sa déclaration
  - Class getReturnType();
  - Class[] getParameterTypes();
  - Class[] getExceptionTypes();
- Invocation
  - Object invoke(Object o, Object args[]);

Les Beans

- Principe
- Les IDEs
- Événements
- Propriétés

[Les Beans]

Principe

- Composants logiciels
  - Assemblés pour faire une application
  - Manipulables dans un IDE
- Respectent
  - Un protocole de communication
  - Un protocole de configuration

[Les Beans]

Les IDEs

- Integrated Development Environment
  - BeanBox
  - JavaStudio
  - Netbeans
  - etc.
- Utilisent :
  - Le modèle d'événements de Java
  - La réflexion
  - La sérialisation
  - Le chargement dynamique

[Les Beans]

Propriétés

- Ce sont des attributs du Bean qui
  - le décrivent, lui ou son état
  - permettent de le contrôler ou de le configurer
- Exemples
  - Valeurs d'entrée d'un additionneur
  - Résultat d'un requête sur un SGBD
  - Couleur de fond d'une fenêtre graphique

[Les Beans / Propriétés]

Mise en œuvre dans le composant

- private <type> <nom>;
- public void set<nom>(<type> valeur);
- public <type> get<nom>();

[Les Beans / Propriétés]

Prise en compte par un IDE

➔L'IDE analyse le Bean pour rechercher les propriétés définies comme indiqué ci-dessus

➔Il fournit alors un éditeur de propriétés (graphique) adapté au type de la propriété qui permet de consulter et de modifier sa valeur

[Les Beans]

Notification d'événement

➔C'est le moyen utilisé par les Beans pour communiquer avec l'extérieur

➔C'est un modèle de type

- fonction réflexe ou *callback*
- observateur ou *Listener*
- source-cible ou *source-target*

[Les Beans / Notification d'événement]

Mise en œuvre dans le composant source

➔Étape 1 : définition des événements significatifs qui seront notifiés

- Un événement porte l'information qu'on souhaite communiquer à l'extérieur
- classe java.util.EventObject
- On étend cette classe pour ajouter les informations pertinentes de l'événement que l'on définit.

Mise en œuvre dans le composant source

- Étape 2 : définition des méthodes de notification à invoquer quand l'événement survient
  - classe java.util.EventListener
  - On étend cette classe pour définir :
    - public interface <nom d'événement>Listener
  - On définit aussi une classe qui implémente l'interface avec des corps de méthode vides. Cela facilite le travail des clients potentiels.

Mise en œuvre dans le composant source

- Etape 3 : on gère l'enregistrement des *listeners* intéressés et leur notification
  - public synchronized void add<XXX>Listener(<XXX>Listener l);
  - public synchronized void remove<XXX>Listener(<XXX>Listener l);
- Classe TooManyListenersException

Quand un événement survient

- Quand un événement survient, le composant source invoque la méthode concernée de tous les *Listeners* enregistrés.

Enregistrement d'un *Listener*

- Un client doit définir un Listener et l'enregistrer auprès du Bean source
  - … l = new <nom d'événement>Listener(…);
  - source.add<nom d'événement>Listener(l);

Performances

[Performances]

SPEC

Standard Performance Evaluation Corporation

http://www.spec.org/

- Organisation non commerciale
- Développe des benchmarks
- Publie les résultats

[Performances / SPEC]

Structure

- OSG : Open System Group
  - Benchmarks niveau composants et niveau système
    - UNIX, NT, VMS
- HPG : High Performance Group
  - Benchmarks dans un environnement numérique
    - Haute performance
- GPC : Graphics Performance Characterization
  - Systèmes graphiques, OpenGL et XWindow

[Performances / SPEC]

SPEC et Java

- Coté Client : SPEC JVM Client98
  - Installé comme une applet
  - Caractéristiques :
    - Measures performance of Java Virtual Machines
    - Applicable to networked and standalone Java client computers, either with disk (e.g., PC, workstation) or without disk (e.g., network computer) executing programs in an ordinary Java platform environment.
    - Requires Java Virtual Machine compatible with JDK 1.1 API, or later
- Coté Serveur : rien pour l'instant

[Performances / SPEC]

Liste des programmes

- _200_check
  - Vérification de fonctionnalités (pas de mesure en sortie)
- _201_compress
  - Variante d'une compression LZW
- _202_jess (Java Expert Shell System)
  - Recherche dans un ensemble de règles croissant
- _209_db
  - Fonctions sur une base de données résidente en mémoire

[Performances / SPEC]

Liste des programmes (suite)

- _213_javac
  - Compilateur Java
- _222_mpegaudio
  - Décompression de fichiers audio
- _227_mtrt
  - Un lancé de rayon avec deux threads pour le rendu
- _228_jack
  - Un générateur d'analyseur syntaxique Java

- Optimisation de programmes Java
  - http://www.javaperformancetuning.com/
- Comparaison de Java avec d'autres langages
  - http://www.bagley.org/~doug/shootout/
  - Août 2001
    - produit de matrice : java = 3.56 g++
    - Tri par tas : java = 2.45 g++

[Performances]

Java Grande

- The Java Grande Forum Benchmark Suite
  - http://www.epcc.ed.ac.uk/javagrande/
- The Java Grande Numeric Working Group
  - http://math.nist.gov/javanumerics/

[Performances / Java Grande]

Composition de la suite

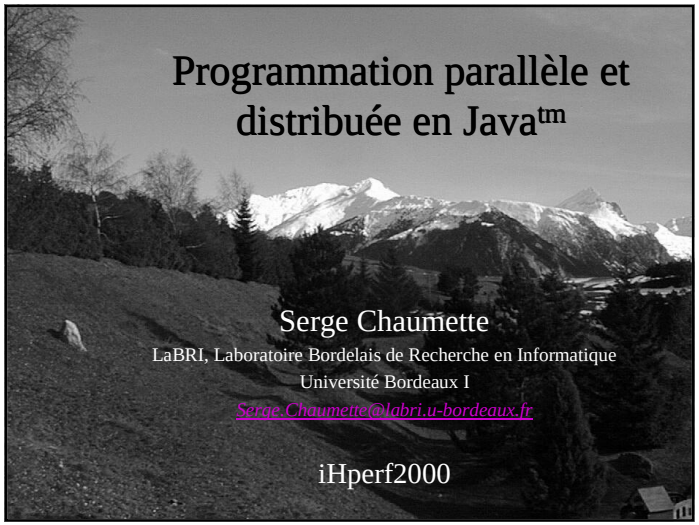
- ***sequential benchmarks***
  - exécution monoprocesseur
- ***multi-threaded benchmarks***
  - exécution parallèle sur multiprocesseurs à mémoire partagée
- ***MPJ benchmarks***
  - exécution parallèle sur multiprocesseurs à mémoire distribuée.
- ***language comparison benchmarks***
  - sous ensemble des *sequential benchmarks* traduits en C

SciMark

Autres benchmarks

- Liste assez complète
  - http://www.epcc.ed.ac.uk/javagrande/links.html
- Plasma
  - http://rsb.info.nih.gov/plasma/
- Linpack
  - http://www.netlib.org/benchmark/linpackjava/
- Drystone
  - http://www.c-creators.co.jp/okayan/
- …

Grids


Programmation parallèle et distribuée en Java™
Serge Chaumette
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.u-bordeaux.fr
iHperf2000