Objets répartis - Partie 1
Le protocole R.M.I.

© Olivier Caron

Polytech'Lille

Plan

- √ Introduction aux objets répartis
- √ Java RMI
- √ CORBA

Les différentes approches existantes (1/2)

√ Les points communs :

- Construire de manière **modulaire** et *simplement* des applications réparties.

Les différentes approches existantes (1/2)

√ Les points communs :

- Construire de manière **modulaire** et *simplement* des applications réparties.
- L'approche objet est un début de solution puisqu'elle apporte encapsulation et modularité.

Les différentes approches existantes (1/2)

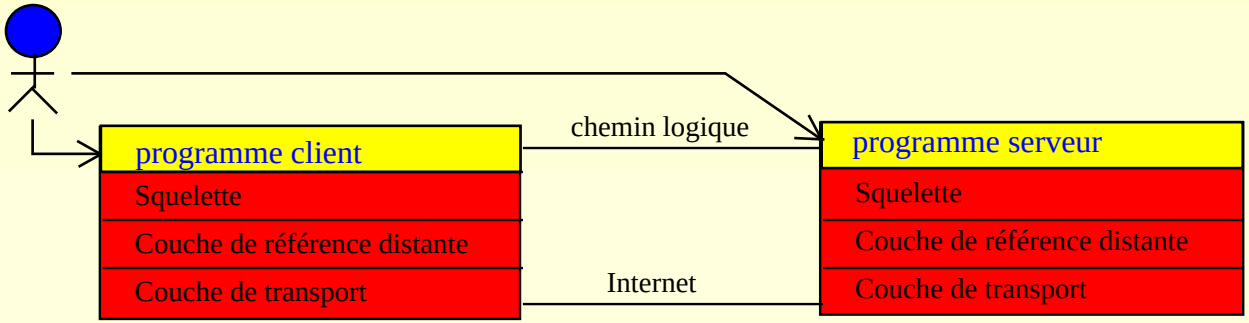
√ Les points communs :

- Construire de manière **modulaire** et *simplement* des applications réparties.
- L'approche objet est un début de solution puisqu'elle apporte encapsulation et modularité.
- il ne reste donc plus qu'à définir un protocole réseau pour faire communiquer des objets entre eux.

Les différentes approches existantes (2/2)

- √ L'approche DCOM de microsoft (ex. OLE)
  réservée au monde Microsoft !
  - ▶ mono-plateforme, multi-langages
- √ La spécification RMI de javasoft.
  - ▶ multi-plateformes, mono-langage (Java)
- √ La norme CORBA (IDL, . . .)
  - ▶ multi-plateformes, multi-langages

Principe de RMI

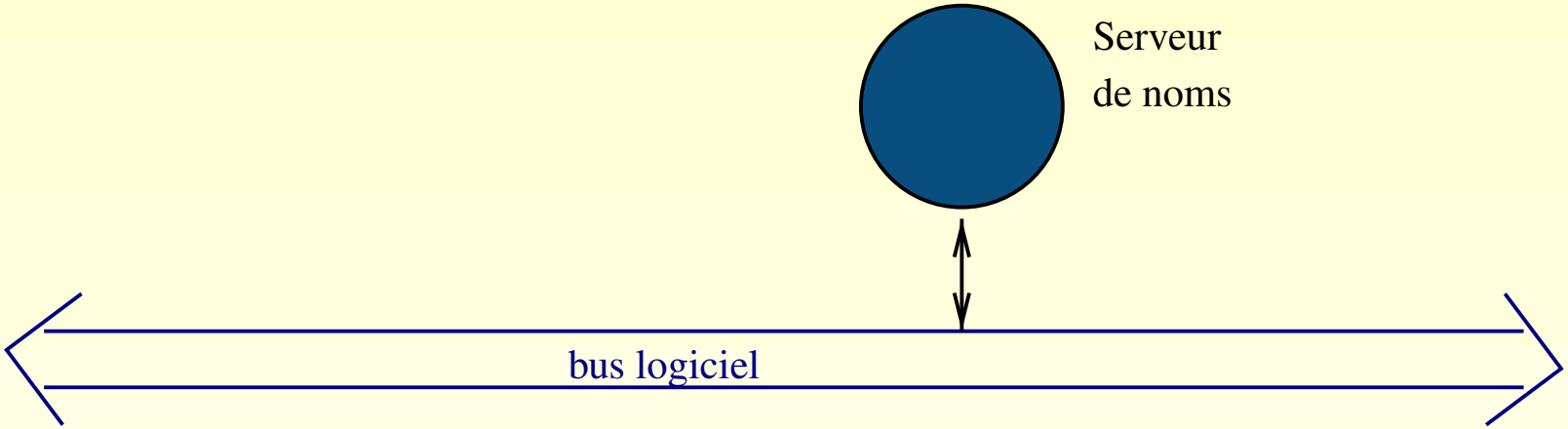


- √ mono-langage : Java
- √ multi-plateformes : Java :-)

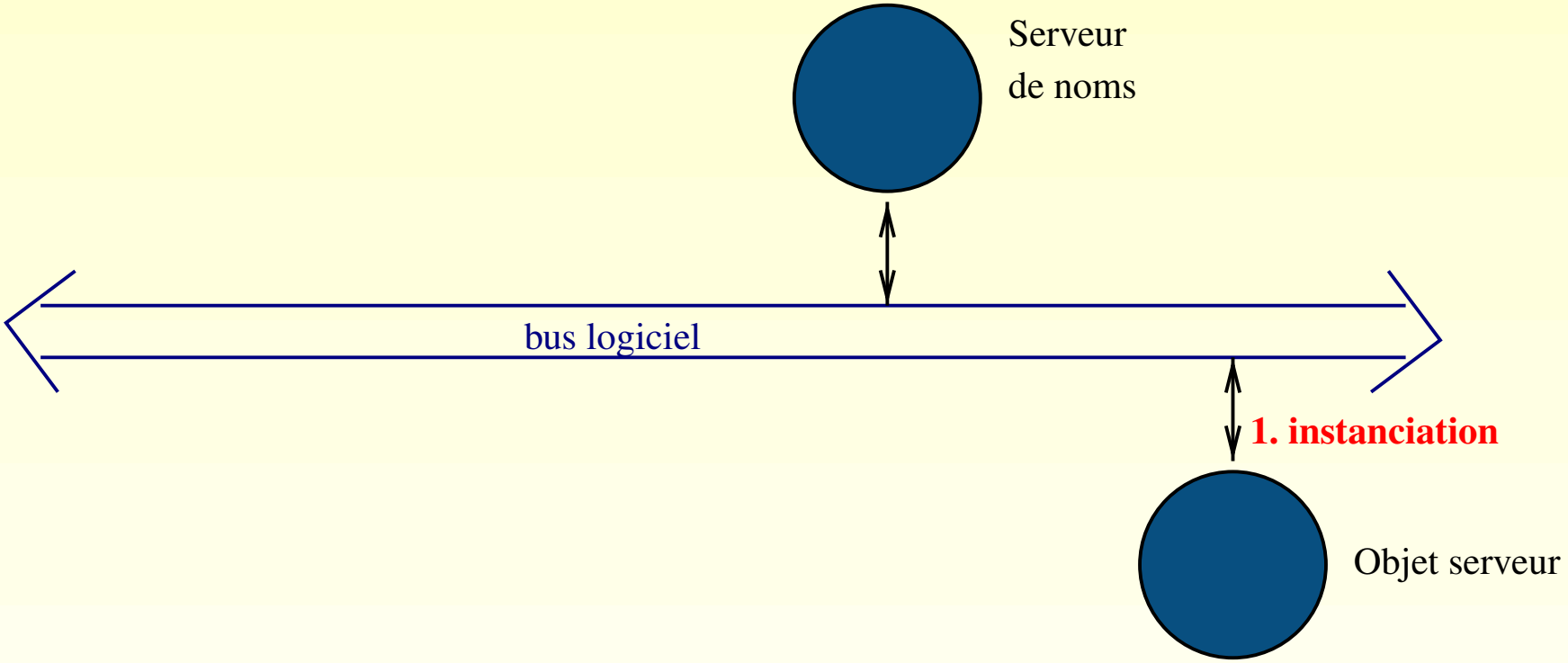
Cycle de vie d'une application répartie

Cycle de vie d'une application répartie

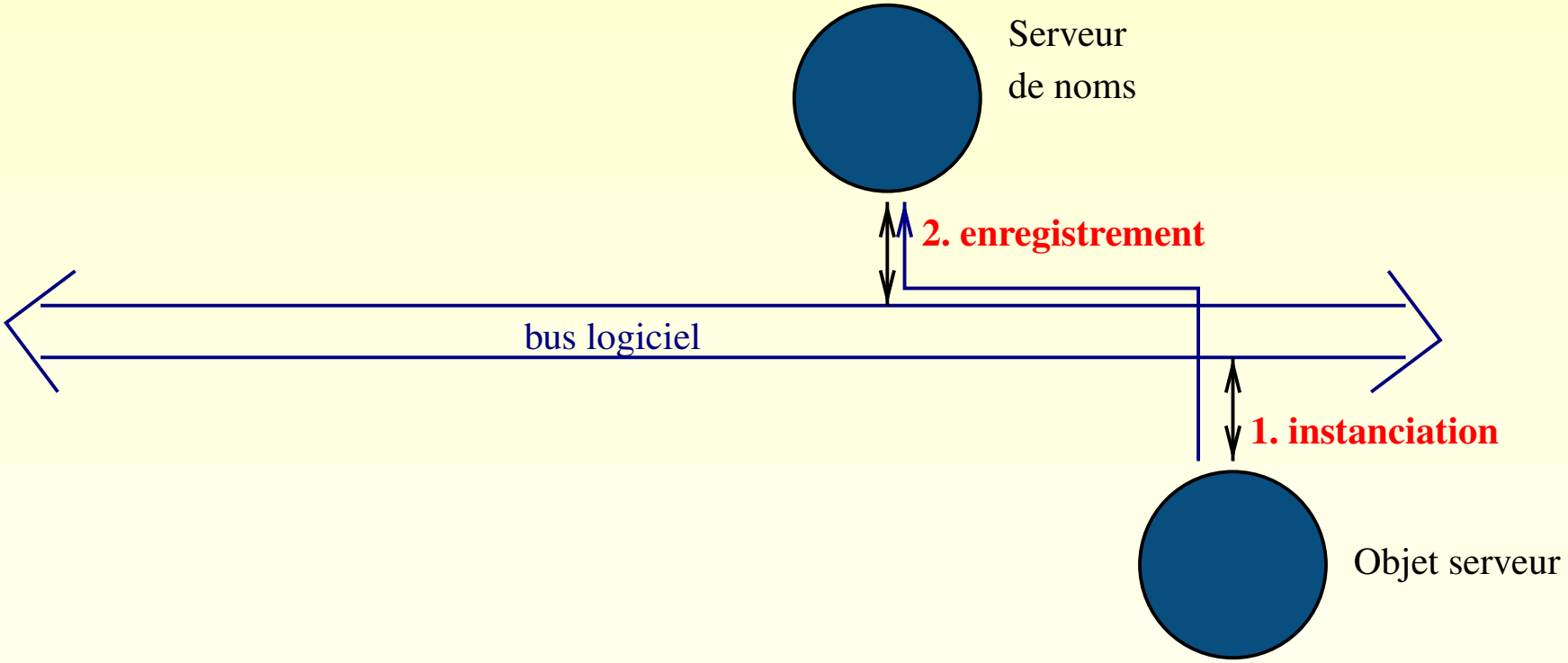
Cycle de vie d'une application répartie



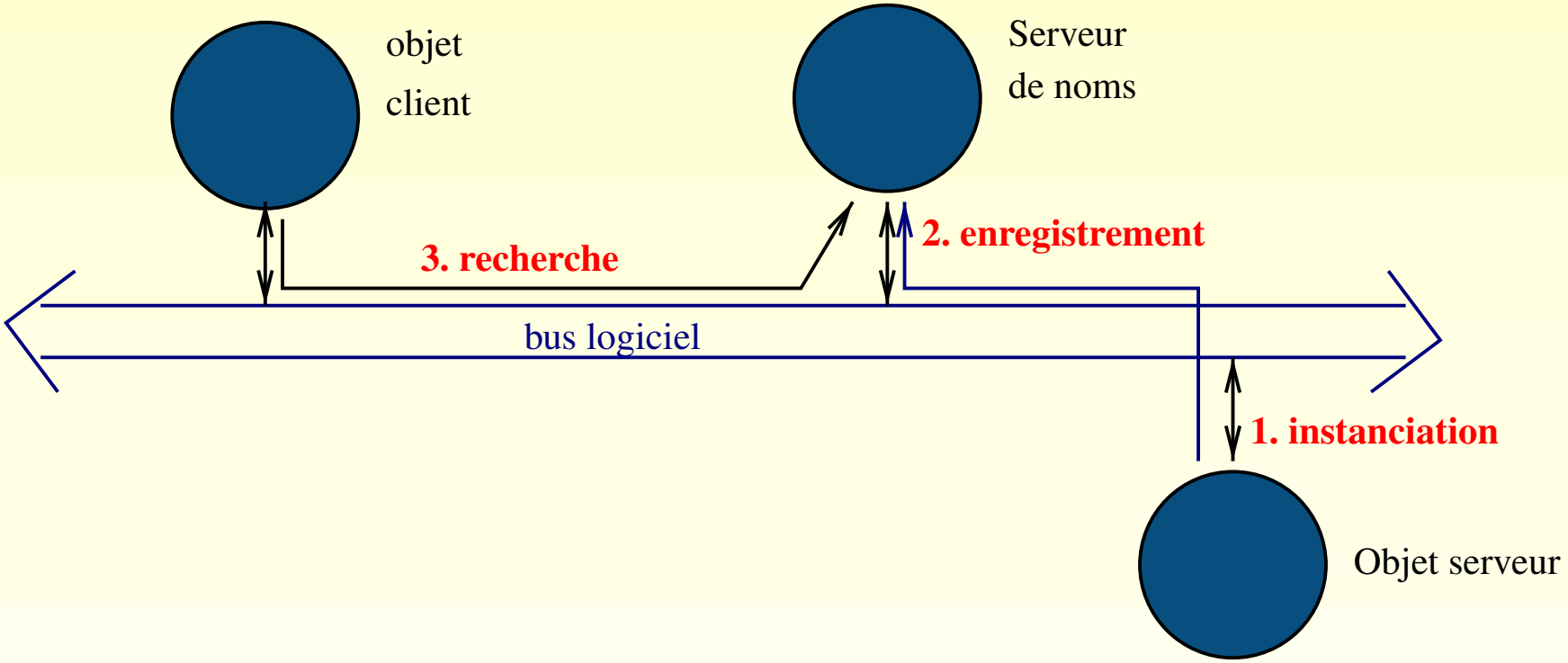
Cycle de vie d'une application répartie



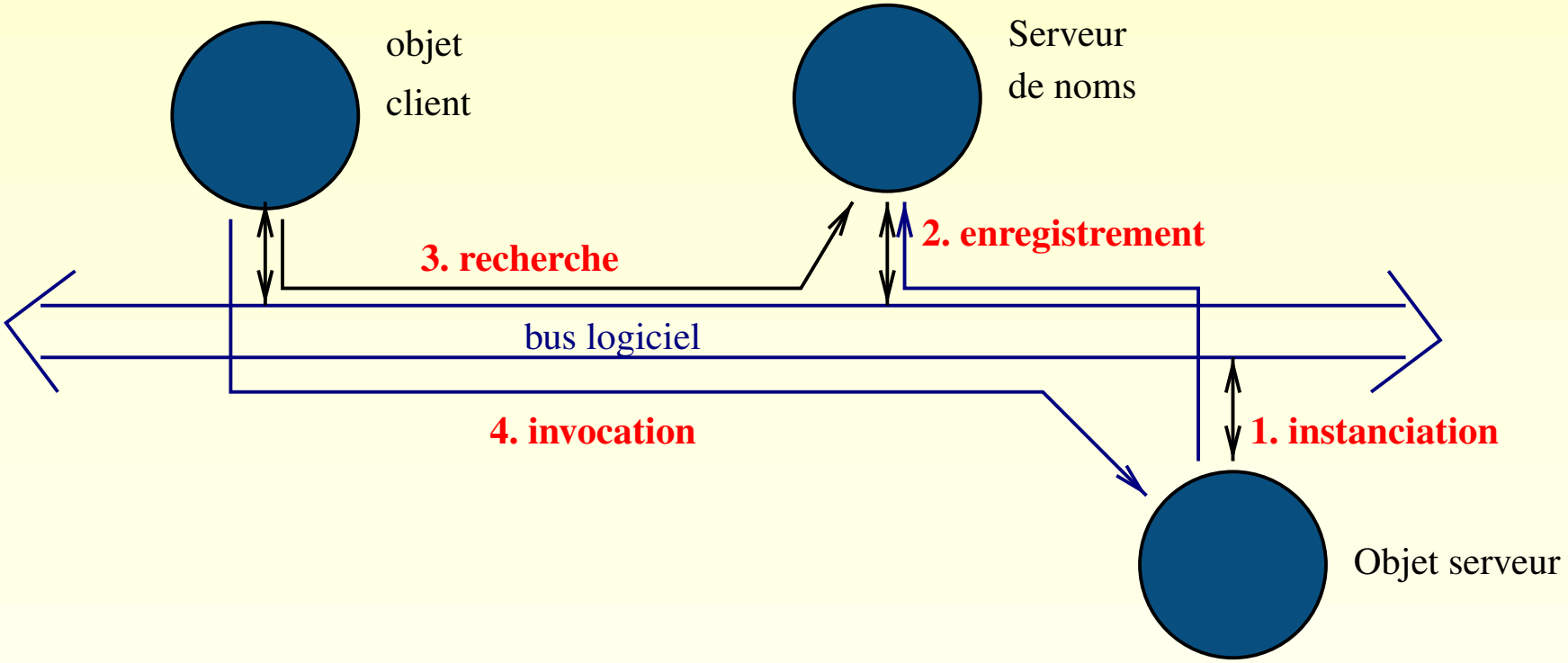
Cycle de vie d'une application répartie



Cycle de vie d'une application répartie



Cycle de vie d'une application répartie



L'application Hello répartie

√ **Définition de l'interface :**

- Le client ne connait l'objet distant QUE par son interface.

L'application Hello répartie

√ **Définition de l'interface :**

- Le client ne connait l'objet distant QUE par son interface.
- permet de cacher certaines méthodes.

L'application Hello répartie

√ **Définition de l'interface :**

- Le client ne connait l'objet distant QUE par son interface.
- permet de cacher certaines méthodes.
- permet d'adapter les objets distants selon les clients (héritage multiple d'interfaces Java)

L'application Hello répartie

√ **Définition de l'interface :**

- Le client ne connait l'objet distant QUE par son interface.
- permet de cacher certaines méthodes.
- permet d'adapter les objets distants selon les clients (héritage multiple d'interfaces Java)

√ Diagramme de séquences UML favorise la définition des interfaces.

Définition de l'interface (1/2)

√ Il est nécessaire de :

- de faire hériter l'interface par l'interface `java.rmi.Remote`

Définition de l'interface (1/2)

√ Il est nécessaire de :

- de faire hériter l'interface par l'interface `java.rmi.Remote`
- de préciser l'exception `java.rmi.RemoteException` pour toutes les méthodes de l'interface.

Définition de l'interface (1/2)

√ Il est nécessaire de :

- de faire hériter l'interface par l'interface `java.rmi.Remote`
- de préciser l'exception `java.rmi.RemoteException` pour toutes les méthodes de l'interface.

Définition de l'interface (2/2)

```
import java.rmi.* ;

public interface HelloInterface extends Remote {
    public String sayHello() throws RemoteException ;
}
```

Construction de l'objet serveur distant (1/2)

√ Il est nécessaire de :

- de faire hériter la classe de `java.rmi.server.UnicastRemoteObject` (gestion réseau)

Construction de l'objet serveur distant (1/2)

√ Il est nécessaire de :

- de faire hériter la classe de `java.rmi.server.UnicastRemoteObject` (gestion réseau)
- d'implémenter les méthodes de (ou des) l'interface

Construction de l'objet serveur distant (1/2)

√ Il est nécessaire de :

- de faire hériter la classe de `java.rmi.server.UnicastRemoteObject` (gestion réseau)
- d'implémenter les méthodes de (ou des) l'interface
- de traiter `java.rmi.remoteException` pour toutes les fonctions (fonction constructeur comprise).

Construction de l'objet serveur distant (2/2)

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

public class Hello extends UnicastRemoteObject
  implements HelloInterface {
    public Hello() throws RemoteException {
    }
    public String sayHello() throws RemoteException {
        return "Hello World!" ;
    }
}
```

Génération des fichiers réseaux

Il est désormais possible de compiler ce programme et de générer les souches réseaux (fichiers stub et squelette) :

```
javac Hello.java     -> genere Hello.class
rmic Hello           -> lecture de Hello.class
                     -> genere Hello_Stub.class
                        et Hello_Skel.class
```

Lancement du serveur

Il faut,au préalable, lancer le registre de noms (rmiregistry) qui se comporte comme un DNS, il associe un nom à un objet.

Le programme va servir à :

√ Créer l'objet

√ Référencer l'objet créé au registre de noms en lui attribuant un nom.

Lancement du serveur

Il faut,au préalable, lancer le registre de noms (rmiregistry) qui se comporte comme un DNS, il associe un nom à un objet.

Le programme va servir à :

√ Créer l'objet

√ Référencer l'objet créé au registre de noms en lui attribuant un nom.

L'API java.rmi.Naming

La classe `java.rmi.Naming` dispose des méthodes suivantes :

méthode	**exceptions**
static void rebind(String nom,Remote obj)	MalformedURLException,RemoteException
static void bind(String nom,Remote obj)	MalformedURLException,RemoteException ,AlreadyBoundException
static void unbind(String nom)	MalformedURLException,RemoteException ,NotBoundException
static Remote lookup(String nom)	MalformedURLException,RemoteException ,NotBoundException
static String [] list(String nom)	MalformedURLException,RemoteException

Le paramètre `nom` désigne l'URL.

Le programme Serveur

```
import java.rmi.* ;
public class Serveur {
    public static void main(String args[]) {
        try {
            Hello h=new Hello() ;
            Naming.rebind("hello",h) ;
            System.out.println("Serveur Hello Pret") ;
        } catch(RemoteException e1) {
            System.out.println("Erreur reseau dans hello") ;
        } catch(java.net.MalformedURLException e2) {
            System.out.println("Erreur adresse reseau dans Hello") ;
        }
    }
}
```

Execution

Le programme Serveur peut être lancé et l'objet distant est accessible à l'adresse URL :

`rmi://nomMachine/hello`

Execution

Le programme Serveur peut être lancé et l'objet distant est accessible à l'adresse URL :

`rmi://nomMachine/hello`

$\surd$ On peut accéder au serveur de noms à distance !

Execution

Le programme Serveur peut être lancé et l'objet distant est accessible à l'adresse URL :

`rmi://nomMachine/hello`

√ On peut accéder au serveur de noms à distance !
√ Pas de hiérarchie dans les noms (CORBA, JNDI)

Execution

Le programme Serveur peut être lancé et l'objet distant est accessible à l'adresse URL :

`rmi://nomMachine/hello`

√ On peut accéder au serveur de noms à distance !
√ Pas de hiérarchie dans les noms (CORBA, JNDI)
√ Pas de courtage (trader CORBA)

Le programme client (1/3)

√ Consultation du serveur de noms (lookup)

Le programme client (1/3)

- √ Consultation du serveur de noms (lookup)
- √ Récupération d'une référence distante

Le programme client (1/3)

- √ Consultation du serveur de noms (lookup)
- √ Récupération d'une référence distante
- √ Utilisation comme un objet local (sauf `RemoteException`)

Le programme client (2/3)

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

public class HelloClient {
    public static void main(String args[]) {
        try {
            System.setSecurityManager(new RMISecurityManager()) ;
            HelloInterface h= (HelloInterface)
              Naming.lookup("rmi://"+args[0]+"/hello") ;
            String message = h.sayHello() ;
            System.out.println(message) ;
```

Le programme client (3/3)

```
        } catch(ArrayIndexOutOfBoundsException e1) {
            System.err.println("erreur Usage: " +
                               " java HelloClient nomServeur") ;
        } catch(NotBoundException e2) {
            System.err.println("hello non référencé")  ;
        } catch(MalformedURLException e3) {
            System.err.println("erreur syntaxe URL") ;
        } catch (RemoteException e4) {
            System.out.println("Erreur accès réseau") ;
        }
    }
}
```

Quelques distinctions (1/3)

√ Comparaison de références distantes impossible !

```
HelloInterface h1 = (HelloInterface)
  Naming.lookup("rmi://"+args[0]+"/hello") ;
HelloInterface h2 = (HelloInterface)
  Naming.lookup("rmi://"+args[0]+"/hello") ;

if (h1==h2) System.out.println("meme objet") ;
```

Quelques distinctions (2/3)

√ Sérialisation des objets ET paramètres.

```
import java.rmi.server.* ;
import java.rmi.* ;
public class Hello extends UnicastRemoteObject
  implements HelloInterface {
    public Hello() throws RemoteException { }
    public String sayHello(Bonjour b) throws RemoteException {
        return b.getBonjour() ;
    }
}
```

```
public class Bonjour implements java.io.Serializable {
   public String getBonjour() { return "bonjour" ; }
}
```

Quelques distinctions (3/3)

- √ Activation à distance impossible sauf :
  - Système d'activation JDK 1.2 (`java.rmi.activation`)

Quelques distinctions (3/3)

√ Activation à distance impossible sauf :

- Système d'activation JDK 1.2 (`java.rmi.activation`)
- Le **pattern** fabrique

La fabrique générique (version base de données)

```
import java.rmi.* ;
public interface TypeObjectFactoryInterface extends Remote {
  public TypeObject create(TypeObjectKey id)
    throws RemoteException, FoundException ;
  public TypeObject findByPrimaryKey(TypeObjectKey id)
    throws RemoteException, NotFoundException ;
  public void destroy(TypeObjectKey id)
    throws RemoteException, NotFoundException ;
}
```

La fabrique `Hello`

```
import java.rmi.* ;

public interface HelloFabriqueInterface extends Remote {
    public HelloInterface create() throws RemoteException ;
}
```

```
import java.rmi.server.* ;
import java.rmi.* ;
public class HelloFabrique extends UnicastRemoteObject
    implements HelloFabriqueInterface {
    public HelloFabrique () throws RemoteException {}
    public HelloInterface create() throws RemoteException {
        return new Hello() ;
    }
}
```

Conclusion

$\surd$ Simple mais efficace.

Conclusion

- √ Simple mais efficace.
- √ Trop simple ? (la classe `Naming`)

Conclusion

- √ Simple mais efficace.
- √ Trop simple ? (la classe `Naming`)
- √ Couplage possible avec CORBA.