I. Les objets distribués

1. Type de passage de paramètres

- Type simple : passage par valeur
- `implements Serializable` : passage par valeur
- `extends Remote` : objet distribué (stub envoyé)

2. Les objets distribués et les *Stubs*

La classe `UnicastRemoteObject` permet de distribuer des objets grâce aux méthodes `exportObject` et `unexportObject`.

Un Stub la référence locale vers un objet distribué. Il se récupère par la méthode `exportObject` :

```
DstribInterf distribObj = new DistribImpl(...);
DstribInterf stub = (DstribInterf) UnicastRemoteObject.exportObject(distribObj, 0);
```

3. Les callbacks

Pour faire du callback, il faut passer un objet distribué en paramètre.

4. Le `rmiregistry`

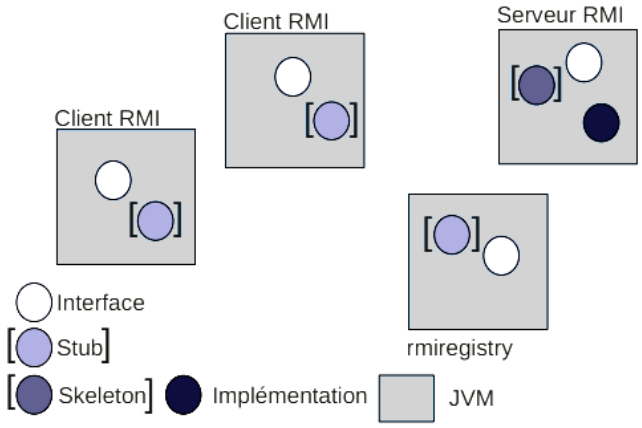
C'est le service de nommage permettant de référencer les objets distribués. Il se lance par :

```
rmiregistry [<port=1099>] -J-Djava.rmi.server.codebase=<http://...|file://...>
```

5. Accès aux sources

Chaque élément (client, serveur, rmiregistry) doit avoir accès à des éléments différents pour pouvoir fonctionner (voir schéma ci-contre).

Ces éléments doivent être accessibles via classpath ou codebase (déclaré par le paramètre `-Djava.rmi.server.codebase=<http://...|file://...>` au lancement de Java).



6. Objets activables

On fait des groupes d'objets, chaque groupe tourne dans une JVM différente.

II. Cycle de développement

- Interface des objets distribués
- Implémentation de l'interface
- Serveur (crée l'objet et le référence au `rmiregistry`)
- Client (récupère l'objet et invoque la méthode voulue)

1. Interface

```
public interface DistribInterf extends Remote {
   public methodeX(...) throws RemoteException;
   ...
}
```

2. Implémentation

```
public class DistribImpl implements DistribInterf, [Serializable] {
      public methodeX(...) {
            ...
      }
}
```

3. Remarque

`rmic` permet de récupérer un skeleton et un stub (inutile pour Java > 1.5), sert pour compatibilité avec ORB.