LOADBALANCER (SOCKETS)

```java
public class LoadBalancer extends Thread {

  static String hosts[] = {"localhost", "localhost"};
  static int ports[] = {8081, 8082};
  static int nbHosts = 2;
  static Random rand = new Random();
  Socket client;

  public LoadBalancer(Socket s) {
    client = s;
  }

  public static void main(String arg[]) {
    try {
      ServerSocket ss = new ServerSocket(8080); // port 8080 par exemple
      while (true) {
        new LoadBalancer(ss.accept()).start();
        // /!\ start et pas pas de run sinon on run sur
        // /!\ le thread principal du coup on bloque
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public void run() {
    try {
      int nb = LoadBalancer.rand.nextInt(LoadBalancer.nbHosts);
      // le serveur à qui on va filer la requête
      Socket server = new Socket(LoadBalancer.hosts[nb], LoadBalancer.ports[nb]);
      InputStream cis = client.getInputStream();
      InputStream sis = server.getInputStream();
      OutputStream cos = client.getOutputStream();
      OutputStream sos = server.getOutputStream();

      /* On fait l'hypothèse qu'on peut tout lire en une seule fois
       * (~ réel sur un réseau local). */
      int nbLus;
      byte[] buff = new byte[1024];
      nbLus = cis.read(buff);
      sos.write(buff, 0, nbLus);
      nbLus = sis.read(buff);
      cos.write(buff, 0, nbLus);
      server.close();
      client.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

CARNET (RMI)

```java
public class CarnetImpl extends UnicastRemoteObject implements Carnet {

  private Map<String, SFiche> fiches;
  private int n;
  public static int nb_carnets = 2;

  public CarnetImpl(int n) throws RemoteException {
    this.fiches = new HashMap<>(); // or any other desired size
    this.n = n;
  }

  @Override
  public void Ajouter(SFiche sf) throws RemoteException {
    fiches.put(sf.getNom(), sf);
    System.out.println("Ajout de la fiche " + sf.getNom());
  }

  @Override
  public RFiche Consulter(String n, boolean forward) throws RemoteException {
    System.out.println("Consulter(" + n + "," + forward + ")");
    SFiche sf = fiches.get(n);

    if (sf == null && forward) {
      for (int i = 1; i <= nb_carnets; i++) {
        if (i != this.n) {
          Carnet carnet;
          try {
            carnet = (Carnet) Naming.lookup("//localhost:4000/Carnet" + i);
              RFiche rf = carnet.Consulter(n, false);
              if (rf != null) {
                return rf;
              }
          } catch (MalformedURLException|RemoteException|NotBoundException e)
            e.printStackTrace();
          }
        }
      }
    }
    if (sf == null) {
      return null;
    }

    return new RFicheImpl(sf);
  }

  public static void main(String args[]) throws RemoteException,
MalformedURLException, AlreadyBoundException {
    LocateRegistry.createRegistry(4000);
    Naming.bind("//localhost:4000/Carnet1", new CarnetImpl(1));
    Naming.bind("//localhost:4000/Carnet2", new CarnetImpl(2));
  }

}
```

```java
public interface Carnet extends Remote {
  public void Ajouter(SFiche sf) throws RemoteException;
  public RFiche Consulter(String n, boolean forward) throws RemoteException;
}

public interface RFiche extends Remote {
  public String getNom() throws RemoteException;
  public String getEmail() throws RemoteException;
}

public class RFicheImpl extends UnicastRemoteObject implements RFiche {
  String nom, email;

  public RFicheImpl(SFiche fiche) throws RemoteException {
    this.email = fiche.getEmail();
    this.nom = fiche.getNom();
  }

  @Override
  public String getNom() throws RemoteException {
    return nom;
  }

  @Override
  public String getEmail() throws RemoteException {
    return email;
  }
}

public interface SFiche extends Serializable {
  public String getNom();
  public String getEmail();
}

public class SFicheImpl implements SFiche {
  String nom, email;

  public SFicheImpl(String nom, String email) {
    this.nom = nom;
    this.email = email;
  }

  @Override
  public String getNom() {
    return this.nom;
  }

  @Override
  public String getEmail() {
    return this.email;
  }
}
```