```
package pipe.cons;

import java.net.*;
import java.io.*;

public class Cons implements IPipeIn {
   private BufferedReader in;
   public Cons(Integer prodPort) throws IOException {
        ServerSocket serverSocket = null;

      try {
         serverSocket = new ServerSocket(prodPort);
      } catch (IOException e) {
         System.err.println("Could not listen on port: " + prodPort);
         System.exit(1);
      }
      Socket clientSocket = null;
      try {
         clientSocket = serverSocket.accept();
      } catch (IOException e) {
         System.err.println("Accept failed.");
         System.exit(1);
      }
      InputStreamReader isr = new
                  InputStreamReader(clientSocket.getInputStream());
      in = new BufferedReader(isr);
   }
   public Object get() throws IOException {
     return in.readLine();
   }
}
```

```
package pipe.cons;
import java.io.IOException;

public interface IPipeIn {

   public Object get() throws IOException;
}
```

```
package pipe.prod;

import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

public class Prod implements IPipeOut {

   private PrintWriter out;

      public Prod(String consAddress, int consPort) throws
UnknownHostException, IOException {
      Socket socket = new Socket(consAddress, consPort);
      out = new PrintWriter(socket.getOutputStream(), true);
     }
   public void put(Object aO) {
     out.println(aO);
   }
}
```

```
package split;

import split.driverIn.DriverIn;
import split.driverIn.IDriverIn;
import split.driverOut.DriverOut;
import split.driverOut.IDriverOut;
import util.ParmsDB;

public class Build {

   private static Logica logica;

   public static void main(String [] args) throws Exception {
      ParmsDB db = new ParmsDB(args);
      build(db);
      logica.run();
   }
   /**
    * Costruisce il distributore
    * @param db of the switches on the command line
    * @throws Exception
    */
   private static void build(ParmsDB db) throws Exception {
      IDriverIn in = new DriverIn(db.getPar("pp")); // port to listen to
producer
      IDriverOut outL = new DriverOut(db.getPar("cla"), db.getPar("clp")); //
low consumer address and port
      IDriverOut outU = new DriverOut(db.getPar("cua"), db.getPar("cup")); //
up consumer address and port
      logica = new Logica(in,outL,outU);
   }
}
```

```
package split.driverIn;
import java.io.IOException;

public interface IDriverIn {
   public String get() throws IOException;
}
```

```
package split.driverIn;
import java.io.IOException;
import pipe.cons.Cons;

public class DriverIn implements IDriverIn {

   Cons in;

   public DriverIn(String pp) throws IOException {
      Integer prodPort = Integer.decode(pp);
      in = new Cons(prodPort);
   }

   public String get() throws IOException {
      return ((String)in.get());
   }

}
```

```
package split.driverOut;

public interface IDriverOut {
   void put(String c);
}
```

```
package split.driverOut;
import java.io.IOException;
import java.net.UnknownHostException;
import pipe.prod.Prod;

public class DriverOut implements IDriverOut {

   Prod out;

   /**
    * @param address
    * @param port
    * @return a DriverOut connected to the in port identified by the previous
switches in db
    * @throws Exception
    * @throws UnknownHostException
    * @throws IOException
    */
   public DriverOut(String address, String port)
   throws Exception, UnknownHostException, IOException {
      out = new Prod(address, Integer.decode(port));
   }
   public void put(String s) {
      out.put(s);
   }
}
```

```
package split;
import java.io.IOException;
import split.driverIn.IDriverIn;
import split.driverOut.IDriverOut;

class Logica {

   private IDriverIn in;
   private IDriverOut outL;
   private IDriverOut outU;

   Logica(IDriverIn in, IDriverOut outL, IDriverOut outU) {
      this.in = in;
      this.outL = outL;
      this.outU = outU;
   }

   void run() throws IOException {
      while (true) {
       outL.put(in.get());
       outU.put(in.get());
      }
   }
}
```

```
package util;
/**
 * TODO implement
 *
 */
public class ParmsDB {
   /**
    * Builds a map from command line switches to the associated arguments
(default value if switch missing)
    *
    * @param args command line arguments
    *
    */
   public ParmsDB(String [] args) {
      return;
   }
   /**
    * @param commandLineSwitch
    *
    * @return the command line argument associated with commandLineSwitch
    */
   public String getPar(String commandLineSwitch) {
      return null;
   }
}
```