Java IO

Esempi

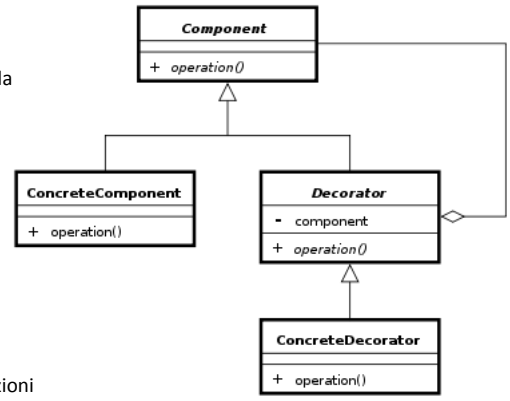
```
BufferedReader br = new BufferedReader(
                        new FileReader("readme.txt"));
br.readLine();
...
```

```
Reader r = new BufferedReader(new FileReader("readme.txt"));
r.read();
…
```

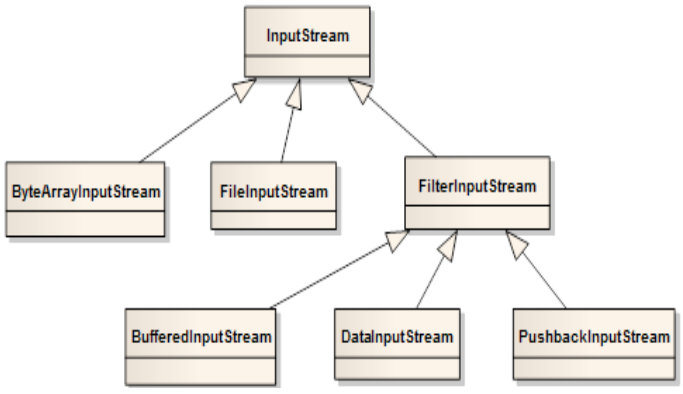
```
…
FileInputStream fis = new FileInputStream("/objects.gz");
BufferedInputStream bis = new BufferedInputStream(fis);
GzipInputStream gis = new GzipInputStream(bis);
ObjectInputStream ois = new ObjectInputStream(gis);
SomeObject someObject = (SomeObject) ois.readObject();
…
```

Il pattern Decorator

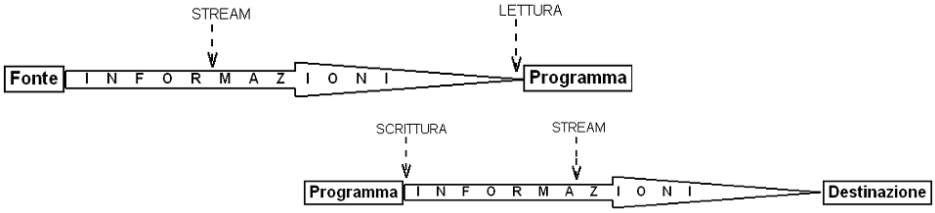
- *Component*:
  - interfaccia (o classe astratta),
  - definisce una o più operazioni astratte da implementare nelle sottoclassi.
- *ConcreteComponent*:
  - una o più classi non astratte
  - implementano Component.
- *Decorator*
  - Solitamente astratto
  - estensione di Component
  - referenzia un Component con un' aggregazione
- *ConcreteDecorator*:
  - è l' implementazione di Decorator,
  - implementa Component e le sue operazioni
  - mantiene un reference verso un Component.



Esempio java.io



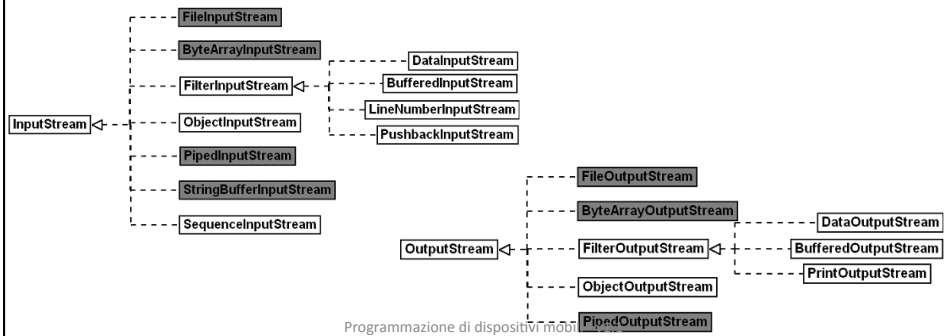
Stream di dati



- Un programma che deve comunicare apre uno STREAM
  - Le informazioni sono trasmesse sequenzialmente!
- Con chi comunica il mio programma?
  - un file, la memoria, un socket
- Gestire uno stream:
  - Aprirlo, leggere o scrivere, chiuderlo

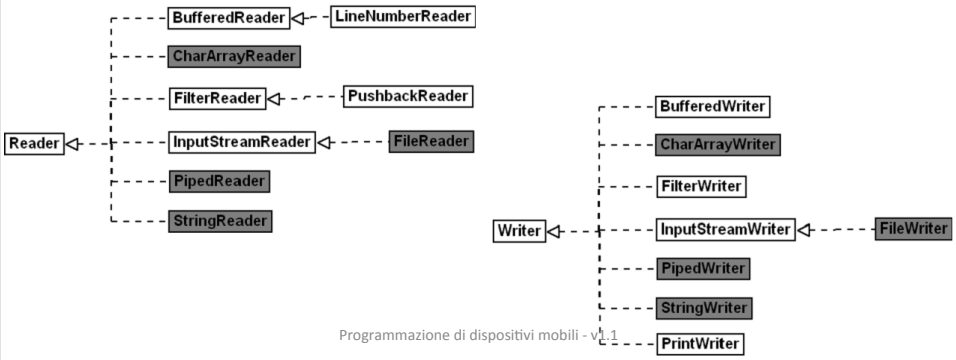
Stream di byte

- InputStream e OutputStream
  - superclassi astratte per i "byte stream"
  - Obbligano le sottoclassi a leggere e scrivere "byte" (8 bit)
  - Hanno ruolo di Component
- Le sottoclassi di Reader (e Writer) implementano stream
  - Alcune sono ConcreteComponent
  - Altre ConcreteDecorator



Stream di caratteri

- Reader e Writer
  - superclassi astratte per i "character stream"
  - Obbligano a leggere e scrivere "char" (16 bit)
  - Hanno ruolo di Component
- Le sottoclassi di Reader (e Writer) implementano stream speciali.
  - Alcune sono ConcreteComponent
  - Altre ConcreteDecorator.



Metodi

InputStream
- int read()throws IOException
- int read(byte cbuf[])throws IOException
- int read(byte cbuf[], int offset, int length) throws IOException

Reader
- int read() throws IOException
- int read(char cbuf[]) throws IOException
- int read(char cbuf[], int offset, int length) throws IOException

- int available() throws IOException :
  - ritorna il numero di byte che possono essere letti all' interno dello stream
- void close() throws IOException
  - rilascia le risorse si sistema associate allo stream
- long skip(long nbytes) throws IOException
  - prova a leggere e a scartare nbytes e ritorna il numero di byte scartati.

OutputStream
- int write(int c)
- int write(byte cbuf[])
- int write(byte cbuf[], int offset, int length)

Writer
- int write(int c)
- int write(char cbuf[])
- int write(char cbuf[], int offset, int length)

Esempio

```
import java.io.*;

public class KeyboardInput {
  public static void main (String args[]) {
    String stringa = null;
    InputStreamReader isr = new
      InputStreamReader(System.in);
    BufferedReader in = new BufferedReader(isr);
    System.out.println("Digita qualcosa e premi " +
      "invio...\nPer terminare il programma " +
      "digitare\"fine\"");
    try {
      stringa = in.readLine();
      while ( stringa != null ) {
        if (stringa.equals("fine")) {
          System.out.println("Programma terminato");
          break;
        }
        System.out.println("Hai scritto: " + stringa);
        stringa = in.readLine();
      }
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      try {
        in.close();
      }
      catch (IOException exc) {
        exc.printStackTrace();
      }
    }
  }
}
```


File

- la classe File astrae il concetto di file generico
  - Anche una directory è un file: un file che contiene altri file.
- Seguono i dettagli dei metodi più interessanti di questa classe:
  - boolean exists() restituisce true se l'oggetto file coincide con un file esistente sul file system
  - String getAbsolutePath() ritorna il path assoluto del file
  - String getCanonicalPath() come il metodo precedente ritorna il path assoluto ma senza utilizzare i simboli "." e ".."
  - String getName() ritorna il nome del file o della directory
  - String getParent() ritorna il nome del file della directory che contiene il file
  - boolean isDirectory() ritorna true se l'oggetto file coincide con una directory esistente sul file system
  - boolean isFile() ritorna true se l'oggetto file coincide con un file esistente sul file system
  - String[] list() ritorna una array di stringhe contenente i nomi dei file contenuti nella directory su cui viene chiamato il metodo. Se questo metodo viene invocato su un file che non è una directory restituisce null
  - boolean delete() tenta di cancellare il file corrente
  - long length() ritorna la lunghezza del file
  - boolean mkdir() tenta la creazione di una directory il cui path è descritto dall'oggetto File corrente
  - boolean renameTo(File newName) tenta di rinominare il file corrente, ritorna true e solo se ha successo
  - boolean canRead() ritorna true se il file o la directory può essere letta dall'utente corrente (ha permesso in lettura)
  - boolean canWrite() ritorna true se il file o la directory può essere può essere modificata
  - boolean createNewFile() crea un nuovo file vuoto come descritto dall'oggetto corrente se tale file non esiste già. Ritorna true se e solo se tale file viene creato

Ancora File

- I costruttori della classe File sono i seguenti:
  - File(String pathname)
  - File(String dir, String subpath)
  - File(File dir, String subpath)

- Esempi

```
File dir = new File("/usr", "local");
File file = new File(dir, "Abc.java");
File dir2 = new File("C:\\directory");
File file2 = new File(dir2, "Abc.java");
```


Esempio

```
import java.io.*;

public class BackupFile {
  public static void main(String[] args) {
    FileInputStream fis = null;
    FileOuputStream fos = null;
    try {
      if (args.length == 0) {
        System.out.println("Specificare nome del file!");
        System.exit(0);
      }
      File inputFile = new File(args[0]);
      File outputFile = new File(args[0]+".backup");
      fis = new FileInputStream (inputFile);
      fos = new FileOuputStream (outputFile);
      int b = 0;
      while ((b = fis.read()) != -1) {
        fos.write(b);
      }
      System.out.println("Eseguito backup in " + args[0]
        + ".backup!");
    } catch (IOException exc) {
      exc.printStackTrace();
    } finally {
      try {
        fis.close();
        fos.close();
      } catch (IOException exc) {
        exc.printStackTrace();
      }
    }
  }
}
```

Serializzare oggetti

- il processo di rendere persistente un oggetto Java salvando lo stato dell'oggetto
  - le variabili d'istanza con i relativi valori
  - tipicamente all'interno di un file.
- Devono implementare l'interfaccia Serializable
  - non contiene metodi
  - Ci dice (per convenzione) se un oggetto è serializzabile
- Se il mio oggetto referenzia oggetti devo serializzare anche loro
  - Se non sono serializzabili non posso serializzare il mio oggetto
  - al runtime otterremmo una java.io.NotSerializableException.
- Le variabili d'istanza marcate con il modificatore transient non vengono serializzata.
  - È obbligatorio marcare transient le variabili non serializzabili
  - Posso usare transient per escudere variabili dalla serializzazione


```
    public Persona(String nome, String cognome, String
cs){
        this.setNome(nome);
        this.setCognome(cognome);
        this.setCodiceSegreto(cs);
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getNome() {
        return nome;
    }
    public void setCognome(String cognome) {
        this.cognome = cognome;
    }

    public String getCognome() {
        return cognome;
    }

    public void setCodiceSegreto (String codiceSegreto) {
        this.codiceSegreto = codiceSegreto;
    }

    public String getCodiceSegreto () {
        return codiceSegreto;
    }

    public String toString() {
        return "Nome: " + getNome() + "\nCognome: " +
          getCognome() + "\nCodice Segreto: " +
          getCodiceSegreto();
    }
}
```

Esempio: Serializzare

```
import java.io.*;

public class SerializeObject {
    public static void main (String args[]) {
        Persona p = new Persona ("Claudio",
          "De Sio Cesari", "xxx");
        try {
            FileOutputStream f = new FileOutputStream (
              new File("persona.ser"));
            ObjectOutputStream s =
              new ObjectOutputStream (f);
            s.writeObject (p);
            s.close ();
            System.out.println("Oggetto serializzato!");
        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
}
```


Esempio

```
import java.io.*;

public class DeSerializeObject {
    public static void main (String args[]) {
        Persona p = null;
        try {
            FileInputStream f =
            new FileInputStream (new
              File("persona.ser"));
            ObjectInputStream s =
            new ObjectInputStream (f);
            p = (Persona)s.readObject();
            s.close ();
            System.out.println("Oggetto " +
              "deserializzato!");
            System.out.println(p);
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }
}
```

Oggetto deserializzato!
Nome: Claudio
Cognome: De Sio Cesari
Codice Segreto: null

Networking

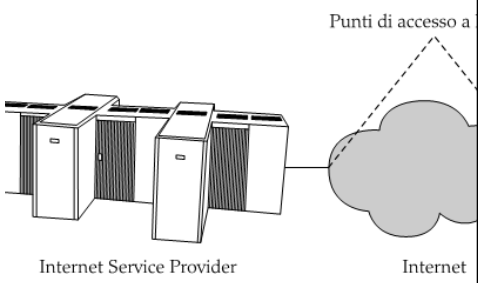
- Da un altro libro

Internet

- Internet: la rete globale che collega tra loro milioni di computer.
  - ogni volta che "navigate" usate Internet
- Internet permette di accedere a numerosi servizi
  - Il World Wide Web
  - Posta elettronica (email)
  - Voip
  - P2P
- A noi interessa Internet come mezzo di comunicazione fra due computer

I protocolli di Internet

- I computer in Internet possono essere connessi l'uno all'altro mediante una grande varietà di mezzi fisici
  - rete ethernet
  - ADSL
  - Modem
- Tutte le trasmissioni trasferiscono di flussi binari lungo la connessione (di rete)
  - dati di applicazioni
  - dati dei protocolli di rete
    - i dati che descrivono come si possa raggiungere il destinatario desiderato e come verificare la presenza di errori e di perdite di dati nella trasmissione.
- I dati dei protocolli seguono regole predeterminate in un particolare protocollo di rete.
- Il protocollo Internet (IP, Internet Protocol), è la base per la connessione di computer sparsi nel mondo.



Indirizzi

- Devo dire l'indirizzo del destinatario
  - computer di destinazione
- Indirizzi IP
  - 130.65.86.66
  - quattro numeri con un numero compreso tra 0 e 255
  - Cioè 32 bit
- Il software di instradamento (routing) distribuito lungo la rete può, quindi, consegnare i dati a B.
- Indirizzi alfanumerici
  - cs.sjsu.edu o java.sun.com.
- Esiste un servizio di traduzione
  - Domain Naming Service (DNS)

Ancora protocolli

- Frammentazione
  - Un unico messaggi grande viene frammentato in piccoli messaggi indipendenti (pacchetti IP)
  - I pacchetti sono numerati e il destinatario li rimette insieme nell'ordine corretto.
- Transmission Control Protocol (TCP)
  - Con IP gestisce una consegna affidabile dei dati
  - Normalmente parliamo di TCP/IP
- TCP ed IP servono a creare dei canali di comunicazione (stream binari) fra computer
  - Fra programmi (applicazioni)

Programmi applicativi

- A quale programma consegno i dati?
  - Devo contrassegnare i dati così da consegnarli al programma giusto.
  - I numeri di porta
    - un numero intero compreso tra 0 e 65535
- Il computer che invia i dati deve conoscere il numero di porta del programma destinatario
- Alcune applicazioni usano numeri di porta "definiti"
  - i server Web - porta 80,
  - POP (Post OfficeProtocol) - porta 110
- I pacchetti TCP, quindi, devono contenere:
  - l'indirizzo Internet del destinatario
  - il numero di porta del destinatario
  - l'indirizzo del mittente
  - il numero di porta del mittente
- Potete pensare a una connessione TCP come a una "conduttura" (pipe) fra due computer che connette insieme due porte

I protocolli applicativi

- Anche gli applicativi quando parlano fra loro seguono delle regole
  - HTTP - Server web e browser
  - SMTP/POP – Client e Server di posta
  - FTP – Client e Server FTP
- Esempio di funzionamneto di una comunicazione Server web-browser
  - Fornisco una URL, Uniform Resource Locator al browser
    - http://java.sun.com/index.html
  - Traduce il nome della macchina remota (DNS)
    - HTTP porta 80
  - Stabilisce una connessione TCP/IP
  - Chiede il file /index.html
    GET /index.html HTTP/1.0
    riga vuota
  - Il server Web riceve la richiesta e la decodifica
  - recupera il file /index.html e lo invia al browser
  - Il browser visualizza il contenuto del file, in modo che lo possiate vedere.

Telnet

- Permette di inviare e ricevere stringhe su una connessione TCP/IP
  - IP - java.sun.com
  - Porta 80 come porta

25

Comandi HTTP

- GET - Richiede una risorsa
- HEAD - Richiede soltanto le informazioni contenute nell'intestazione di una risorsa
- OPTIONS - Richiede le opzioni di comunicazione di una risorsa
- POST - Fornisce dati in ingresso per un comando eseguito sul server e ne ottiene il risultato
- PUT - Memorizza una risorsa sul server
- DELETE - Cancella una risorsa dal server
- TRACE - Conserva un tracciato della comunicazione con il server

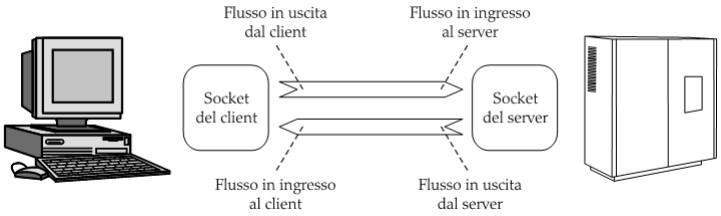
SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 headers and 4 lines in the body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye

POP

S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK mrose's maildrop has 2 messages (320 octets)
C: STAT S: +OK 2 320
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends message 1>
S: .
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends message 2>
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)

Un programma client



- Socket s = new Socket(nomeHost, numeroPorta);
  - Esempio
    final int HTTP_PORT = 80;
    Socket s = new Socket("java.sun.com", HTTP_PORT);
  - Il costruttore di socket lancia un'eccezione di tipo UnknownHostException se non riesce a trovare il computer a cui connettersi.
- Un socket è un oggetto che incapsula una connessione TCP/IP. Per comunicare con l'altra estremità della connessione, usate iflussi di ingresso e uscita connessi al socket.

Stream del socket

- ottenere i flussi entranti e uscenti (binari)
  - InputStream in = s.getInputStream();
  - OutputStream out = s.getOutputStream();

- Per inviare e ricevere testo, dovreste decorare:

```
BufferedReader reader = new BufferedReader(new InputStreamReader(in));

PrintWriter writer = new PrintWriter(out);
 writer.print(“ Una stringa”);
oppure
BufferedWriter writer = new BufferedWriter (new OutputStreamWriter(out));
writer.write(“Una stringa”);
```

- Ricordatevi di inviare i dati

```
writer.flush();
```


WebGET

- Esempio nel Libro

Simple Client

```
import java.net.*;
import java.io.*;
public class SimpleClient {
  public static void main(String args[]) {
    try {
      String host = "127.0.0.1";
      if (args.length != 0) {
         host = args[0];
      }
      Socket s = new Socket(host, 9999);
      BufferedReader br = new BufferedReader(new
        InputStreamReader(s.getInputStream()));
      System.out.println(br.readLine());
      br.close();
      s.close();
    } catch (ConnectException connExc) {
      System.err.println("Non riesco a connettermi "
        + "al server");
    } catch (IOException e) {
      System.err.println("Problemi...");
    }
  }
}
```


Java URL

- Java contiene anche una classe URLConnection, che fornisce un comodo supporto per il protocollo HTTP.
  - La classe URLConnection si occupa della connessione al socket,
  - è anche in grado di gestire il protocollo FTP (File Transfer Protocol).
- Uso delle URL Java

  URL u = new URL("http://java.sun.com/index.html");
  URLConnection connection = u.openConnection();
- ottenere un flusso di ingresso:

```
InputStream in = connection.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(in));

boolean done = false;
while (!done)
{
  String input = reader.readLine();
  if (input == null)
    done = true;
  else fa qualcosa con la riga appena letta
}
```

HttpURLConnection

- Gestione delle proprietà delle richieste
  - If-Modified-Since
    - connection.setIfModifiedSince(data);

```
GET risorsa HTTP/1.0
If-Modified-Since: data e ora
riga vuota
```

- Gestione delle risposte
  - Parsing dell'header

```
HTTP/1.1 200 OK
Date: Sun, 29 Aug 1999 00:15:48 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Thu, 24 Jun 1999 20:53:38 GMT
Content-Length: 4813
Content-Type: text/html
riga vuota
dati richiesti
```

```
HttpURLConnection httpConnection = (HttpURLConnection) connection;
int code = httpConnection.getResponseCode(); // es. 404
String message = httpConnection.getResponseMessage(); // es. "Not Found"
…
int length = connection.getContentLength();
String type = connection.getContentType();
```

Server socket

- Un programma server attende che i client si connettano a un particolare porta
  - È in ascolto di connessioni entranti

```
ServerSocket server = new ServerSocket(8888);
Socket s = server.accept();
```

- Il programma è fermo ed aspetta una connessione
  - Se arriva assegna la variabile s e continua
  - Da s ottengo i flussi di ingresso e uscita

```
BufferedReader in = new BufferedReader(new
    InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(s.getOutputStream());
```


Simple Server

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

public class SimpleServer {
  public static void main(String args[]) {
    ServerSocket s = null;
    try {
      s = new ServerSocket(9999);
      System.out.println("Server avviato, in ascolto"
        + " sulla porta 9999");
    } catch (IOException e) {
      e.printStackTrace();
    }
    while (true) {
      try {
        Socket s1 = s.accept();
        OutputStream slout = s1.getOutputStream();
        BufferedWriter bw = new BufferedWriter(new
          OutputStreamWriter(slout));
        bw.write("Ciao client sono il server!");
          System.out.println("Messaggio spedito a " +
          s1.getInetAddress());
        bw.close();
        s1.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Esempio SMTP

SMTP

…

SMTP

S: 220 smtp.example.com ESMTP Postfix

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 headers and 4 lines in the body.
C: Your friend,
C: Bob
C: .

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 headers and 4 lines in the body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 headers and 4 lines in the body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 headers and 4 lines in the body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT

SMTP

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 headers and 4 lines in the body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye

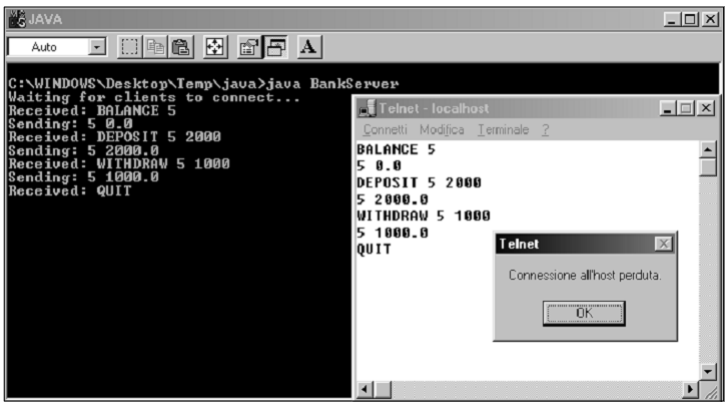
Sviluppare server

- Definire un protocollo al livello applicazione
  - I client devono usare tale protocollo per ottenere un servizio
- Esempio di protocollo

Tabella 2 Un semplice protocollo per l'accesso a una banca

Richiesta del client	Risposta del server	Significato
`BALANCE` *n*	*n* e il saldo	Leggi il saldo del conto *n*
`DEPOSIT` *n a*	*n* e il nuovo saldo	Versa la somma *a* nel conto *n*
`WITHDRAW` *n a*	*n* e il nuovo saldo	Preleva la somma *a* dal conto *n*
`QUIT`	Nessuna	Termina la connessione

Il BankServer



Consigli per il progetto

- Passo 1. Determinate se ha veramente senso realizzare un server a sé stante e un corrispondente client
- Passo 2. Progettate un protocollo di comunicazione
  - Individuate i messaggi che si scambiano il client e il server
    - Sia le risposte in caso di errore o in caso di successo.
  - Per ogni richiesta e risposta, chiedetevi come venga indicata la fine dei dati.
    - I dati sono contenuti su un' unica riga? I dati possono essere terminati da una riga speciale? Chi invia i dati chiude il socket al termine?
  - Per la comunicazione tra client e server usate il formato di testo
- Passo 3. Realizzate il programma server
  - Il server aspetta richieste di connessione attraverso un socket e le accetta. Quindi,riceve comandi, li interpreta, e invia una risposta al client.
- Passo 4. Collaudate il server usando il programma telnet
  - Provate tutti i comandi del protocollo di comunicazione.
- Passo 5. Quando il server funziona, scrivete un programma client
  - Il programma client interagisce con l' utente, ne trasforma le richieste in comandi del protocollo, invia i comandi al server, riceve la risposta e la visualizza per l' utente.