Socket

Programmazione Distribuita - A.A. 2020/2021

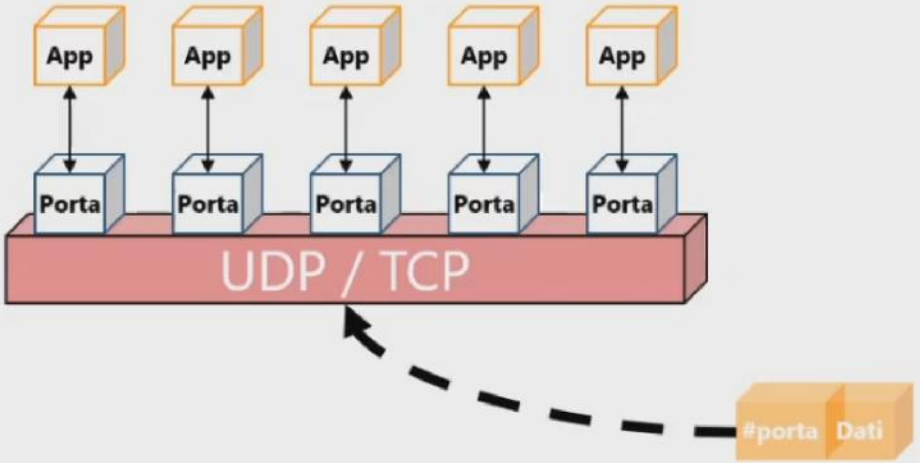
Biagio Cosenza
Dipartimento di Informatica
Università di Salerno
http://cosenza.eu/
bcosenza@unisa.it

Organizzazione della Lezione

- Programmazione con i socket
  - Socket TCP
  - Stream
- Alcuni esempi di uso dei socket
  - Hello World
  - Un registro di nomi con architettura client-server
- Conclusioni

Richiami di TCP: Porte

- I protocolli TCP e UDP usano le porte per mappare i dati in ingresso con un particolare processo attivo su un computer
- Ogni socket è legato a un numero di porta così che il livello TCP può identificare l'applicazione a cui i dati devono essere inviati



Socket: Definizione

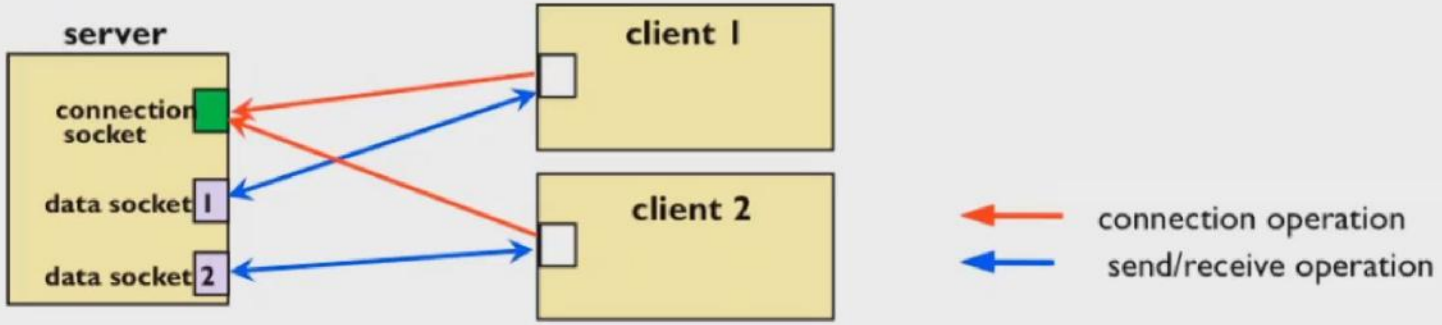
- A livello programmatico, un socket è definito come un «identificativo univoco che rappresenta un canale di comunicazione attraverso cui l'informazione è trasmessa» [RFC 147]
- La comunicazione basata su socket è indipendente dal linguaggio di programmazione
- Client e server devono concordare solo su protocollo (TCP o UDP) e numero di porta

I Socket TCP in Java

- Java fornisce le API per i socket in `java.net`
- Socket TCP:
  - astrazione fornita dal software di rete
  - permette di ricevere e trasmettere flussi dati
- Comunicazione bidirezionale
- Socket come endpoint (per il programmatore)
  - caratterizzato da un indirizzo IP e da una porta
- Client-server
  - naturale suddivisione dei compiti
  - chi in attesa di connessioni (server) e chi cerca di connettersi ad un servizio
  - asimmetrica

Come funzionano I socket

- I socket su stream sono supportati da due classi:
  - `ServerSocket`: per accettare connessioni (socket di connessione)
  - `Socket`: per scambio di dati (socket di dati)

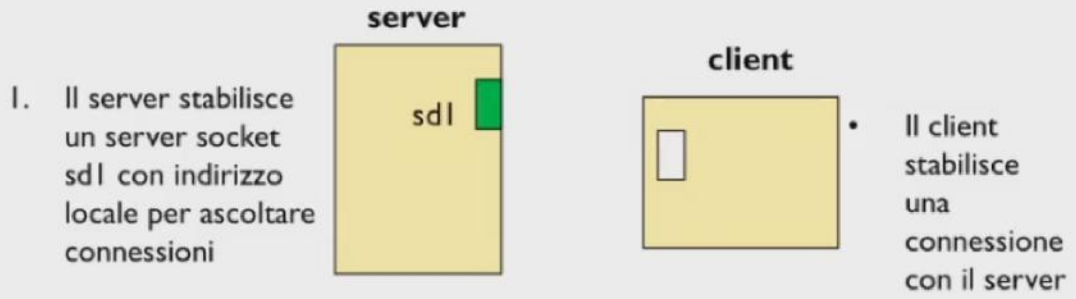


Metodi di `ServerSocket`

- `ServerSocket(int port)`
  - crea un server socket su una specifica porta
- `Socket accept()throws IOException`
  - aspetta connessioni su questo socket e le accetta
  - metodo bloccante fino a quando non viene fatta una connessione
  - un nuovo Socket viene creato e restituito
- `public void close() throws IOException`
  - chiude il socket
- `void setSoTimeout(int timeout)throws SocketException`
  - setta un timeout (in ms) per una call ad accept
  - se il tempo passa senza una connessione, viene lanciata una eccezione `java.io.InterruptedIOException`

Costruzione di uno stream Socket (1)

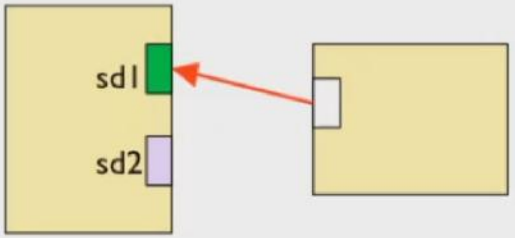
- `sd1` – socket di ascolto



Costruzione di uno stream Socket (2)

- `sd2` – socket di connessione
- usato per lo scambio dei dati con un client

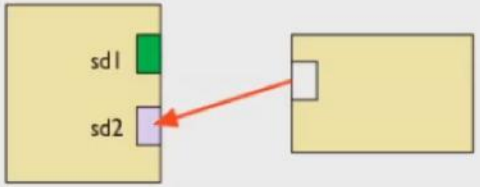
2. Il server accetta una connessione e crea un nuovo socket per dati sd2



Costruzione di uno stream Socket (3)

- Scambio di dati sul socket di connessione `sd2`
- Il client invia dati

3. Il server effettua una receive

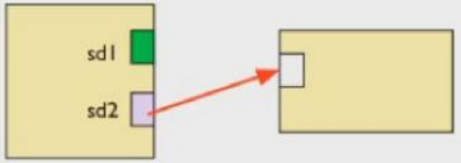


- Il client effettua una send

Costruzione di uno stream Socket (4)

- Scambio di dati sul socket di connessione `sd2`
- Il server invia dati

4. Il server invia la risposta

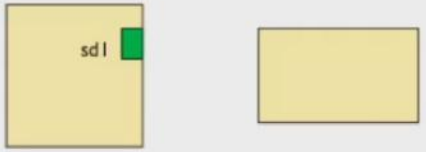


- Il client effettua una receive

Costruzione di uno stream Socket (5)

- Il server chiude la connessione

5. Quando il protocollo termina, il server chiude ed elimina il socket sd2



- Il client chiude il socket

Stream

- Gli stream sono una maniera per prendere dati e trasferirli da qualche altra parte
  - *"Prendi dati, fanne qualcosa, memorizza il risultato"*
  - i device sono spesso trattati come stream
- Per usare un device
  - fare setup/creare di un oggetto che rappresenta dove (o da dove) saranno inviati (o ricevuti) i dati, per poi inviare (ricevere) i dati
- Esempi:
  - un file viene visto come uno stream, un socket viene visto come uno stream, etc.
- Lo schermo non è uno stream:
  - l'utente lo vede per intero
  - si aspetta di poterlo manipolare (insert/remove)
  - l'adiacenza indica relazioni su più dimensioni (righe e colonne)

Stream in Java

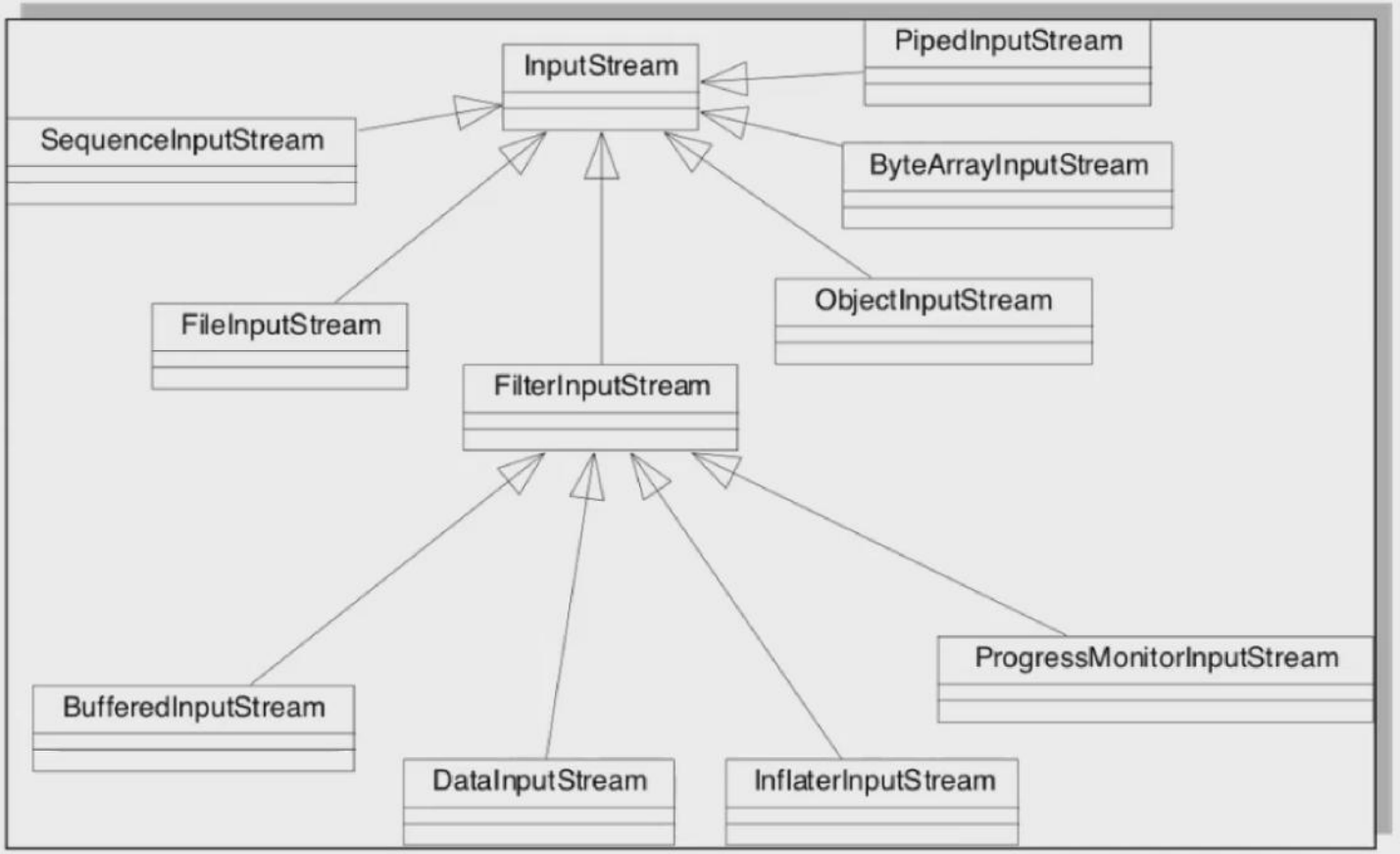
- Uno stream è una sequenza ordinata di byte
  - `java.io.InputStream`:
    dati che da una sorgente esterna possono essere usati dal programma

```
int read()
int read(byte[]b)
int read(byte[] b,int off,int len)
void mark(int readlimit)
boolean markSupported()
void reset()
void close()
int available()
long skip(long n)
```

  - `java.io.OutputStream`:
    dati che il programma può inviare

```
void close()
void flush()
void write(byte[]b)
void write(byte[]b,int off,int len)
void write(int b)
```

Input Stream in Java



Stream di Input e Output

- Gli stream sono sempre presenti in coppia
  - uno sa leggere (dallo stream) e l'altro sa scrivere (sullo stream)
- Per ogni tipo di `InputStream` (tranne `SequenceInputStream`) c'è il corrispondente `OutputStream` associato
  - `FileInputStream` ⇒ `FileOutputStream`
  - `DataInputStream` ⇒ `DataOutputStream`

Esempio di uso dei socket

- Hello World

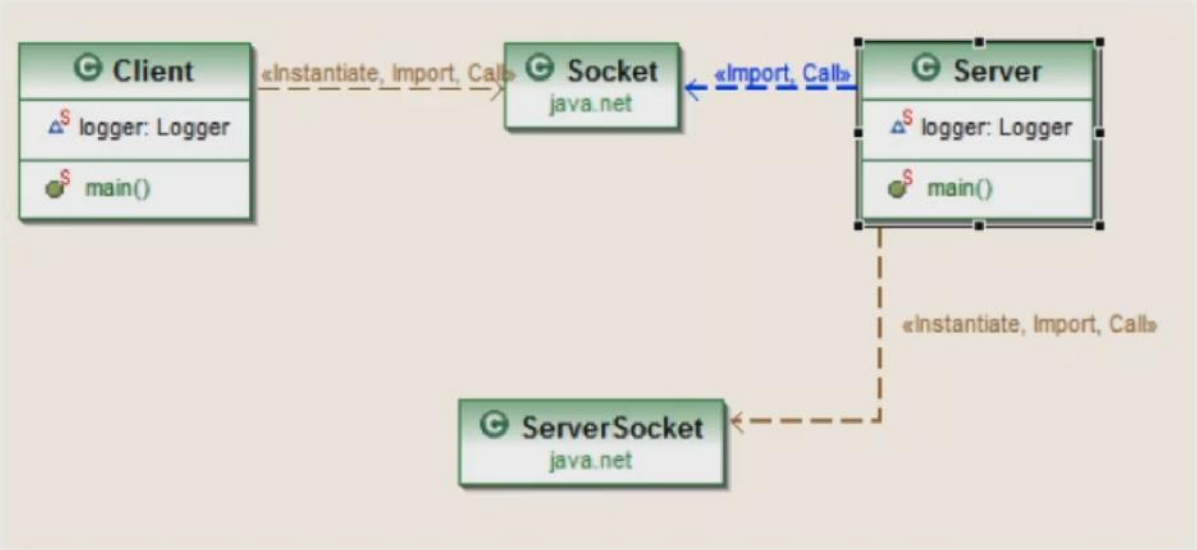


Diagramma delle classi

HelloWorld Client

- Import
- Definizione classe
- Logger
- Socket connessione
- Stream output
- Stream input
- Invia una stringa e riceve un `Object`
- Chiusura socket
- Gestione delle eccezioni

```
import java.io.*;
import java.net.*;
import java.util.logging.Logger;
public class Client {
 static Logger logger = Logger.getLogger("global");

 public static void main(String args[]) {
  try{
   Socket socket = new Socket ("localhost", 9000);
   ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
   ObjectInputStream in = new  ObjectInputStream(socket.getInputStream());
   out.writeObject("Giovanni");
   System.out.println(in.readObject());
   socket.close();
  } catch(EOFException e) {
   logger.severe("Problemi con la connessione:"+  e.getMessage());
   e.printStackTrace();
  } catch(Throwable t) {
   logger.severe("Lanciata Throwable:"+ t.getMessage());
   t.printStackTrace();
  }
 }
}
```

HelloWorld Server (1)

```
import java.io. *;
import java.net. *;
import java.util.logging.Logger;

public class Server {
 static Logger logger = Logger.getLogger("global");

 public static void main(String[] args) {
  try{
   ServerSocket serverSocket = new ServerSocket(9000);
   logger.info("Socketok,accetto conn...");

  Socket socket = serverSocket.accept();
   logger.info("Accettata una connessione....");

   ObjectOutputStream oS = new
        ObjectOutputStream(socket.getOutputStream());
   ObjectInputStream iS = new ObjectInputStream(socket.getInputStream());

   //...
```

HelloWorld Server (2)

- riceve la stringa
- risponde salutando il client
- gestione eccezioni
  - di connessione
  - altre eccezioni

```
  //...

  String nome= (String) iS.readObject();
  logger.info("Ricevuto:"+ nome);

  oS.writeObject("Hello"+ nome);
  socket.close();

 } catch(EOFException e) {
   logger.severe("Problemi con la connessione:"+ e.getMessage());
   e.printStackTrace();
 } catch(Throwable t) {
   logger.severe("Lanciata Throwable:"+ t.getMessage());
   t.printStackTrace();
 }
}
}
```

HelloWorld Server (1)

- Import
- Definizione classe
- main
- crea un `ServerSocket`
- accetta connessioni
- quando c'e' un client
  - prende l'output stream
  - e l'input stream

```
import java.io. *;
import java.net. *;
import java.util.logging.Logger;

public class Server {
 static Logger logger = Logger.getLogger("global");

 public static void main(String[] args) {
  try{
   ServerSocket serverSocket = new ServerSocket(9000);
   logger.info("Socketok,accetto conn...");

  Socket socket = serverSocket.accept();
   logger.info("Accettata una connessione....");

   ObjectOutputStream oS = new
        ObjectOutputStream(socket.getOutputStream());
   ObjectInputStream iS = new ObjectInputStream(socket.getInputStream());

   //...
```

HelloWorld Server (2)

- riceve la stringa
- risponde salutando il client
- gestione eccezioni
  - di connessione
  - altre eccezioni

```
  //...

  String nome= (String) iS.readObject();
  logger.info("Ricevuto:"+ nome);

  oS.writeObject("Hello"+ nome);
  socket.close();

 } catch(EOFException e) {
   logger.severe("Problemi con la connessione:"+ e.getMessage());
   e.printStackTrace();
 } catch(Throwable t) {
   logger.severe("Lanciata Throwable:"+ t.getMessage());
   t.printStackTrace();
 }
}
}
```

Esempio: Registro di nomi con architettura client-server

- Un esempio dalla struttura semplice
- Il server
  - mantiene dei record
    - composti da nome e indirizzo
- Il client può
  - inserire un record: passando un oggetto da inserire
    - record verrà memorizzato
  - ricercare un record: passando un oggetto solamente con il campo nome inserito
    - ricerca di un record, restituzione di un valore

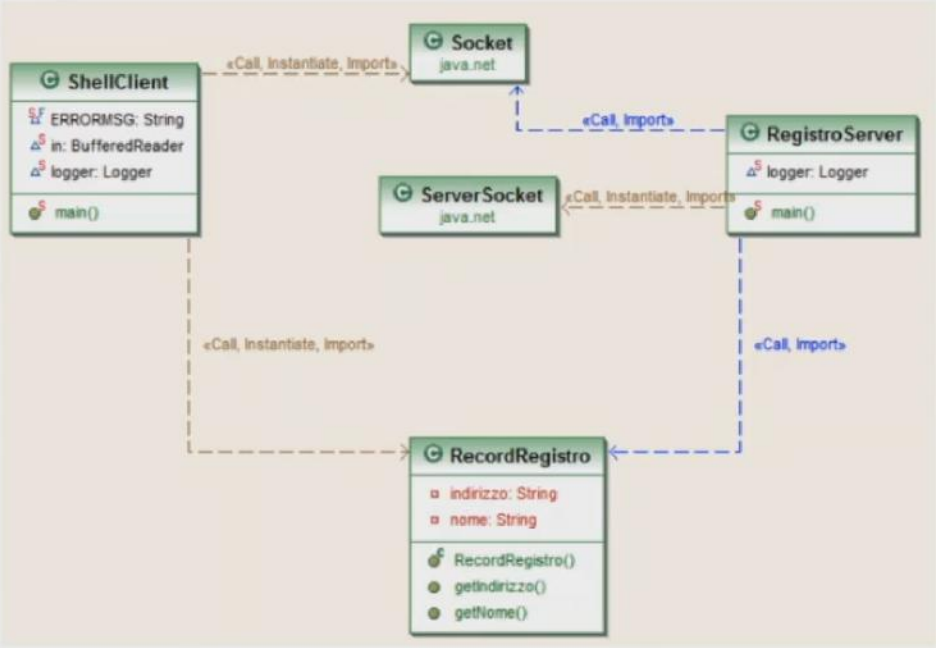
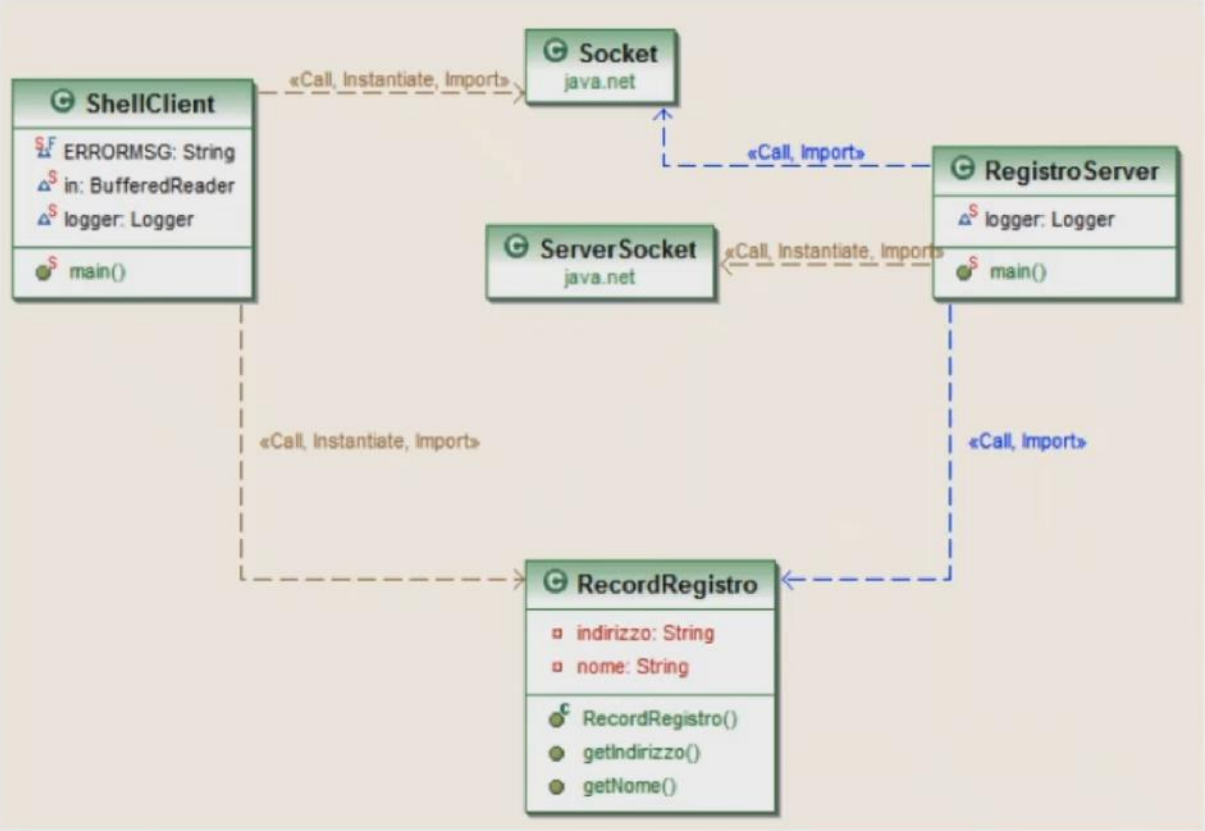


Diagramma delle classi

Esempio: Registro di nomi con architettura client-server

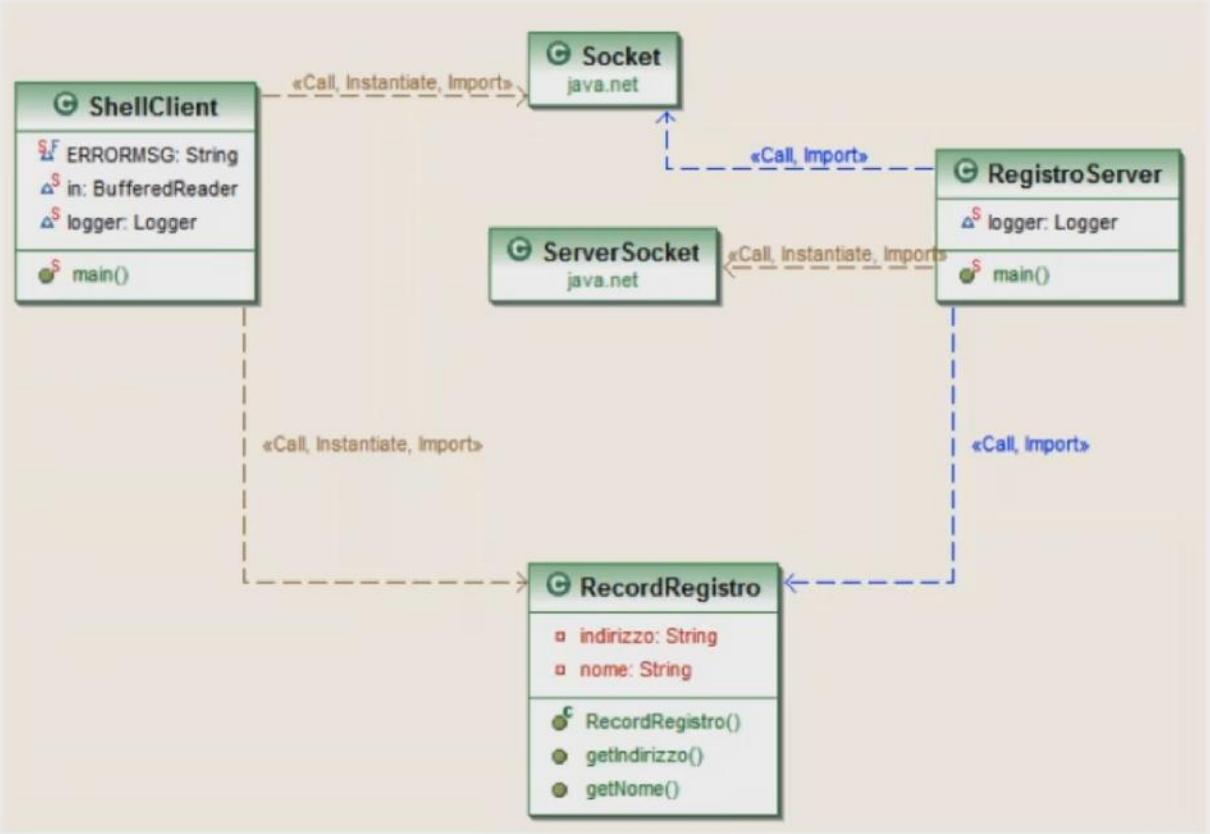


RecordRegistro

- import
- deve essere trasmessa sulla rete
- versione della classe
- costruttore
- il resto della classe non merita commenti

```
import java.io.Serializable;

public class RecordRegistro implements Serializable {
   private static final long serialVersionUID = -4147133786465982122L;

   public RecordRegistro(String n, String i) {
      nome = n;
      indirizzo = i;
   }

   public String getNome() {
     return nome;
   }

   public String getIndirizzo() {
     return indirizzo;
   }

   private String nome;
   private String indirizzo;
}
```

La Classe RegistroServer



RegistroServer (1)

- Import
- Classe Logger
- Mantiene le coppie nome, valore
- Socket di connessione Ciclo
- accetta connessioni
  - crea stream input
  - legge l'oggetto (bloccante)

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.logging.Logger;

public class RegistroServer {
 static Logger logger = Logger.getLogger("global");

 public static void main(String[] args) {
  HashMap<String, RecordRegistro> hash = new HashMap<String,RecordRegistro>();
  Socket socket = null;
  System.out.println ("In attesa...");
  try{
   ServerSocket serverSocket = new ServerSocket(7000);

   while(true) {
    socket = serverSocket.accept();
    ObjectInputStream inStream = new ObjectInputStream(socket.getInputStream());
    RecordRegistro record = (RecordRegistro)inStream.readObject();

    //...
```

RegistroServer (1)

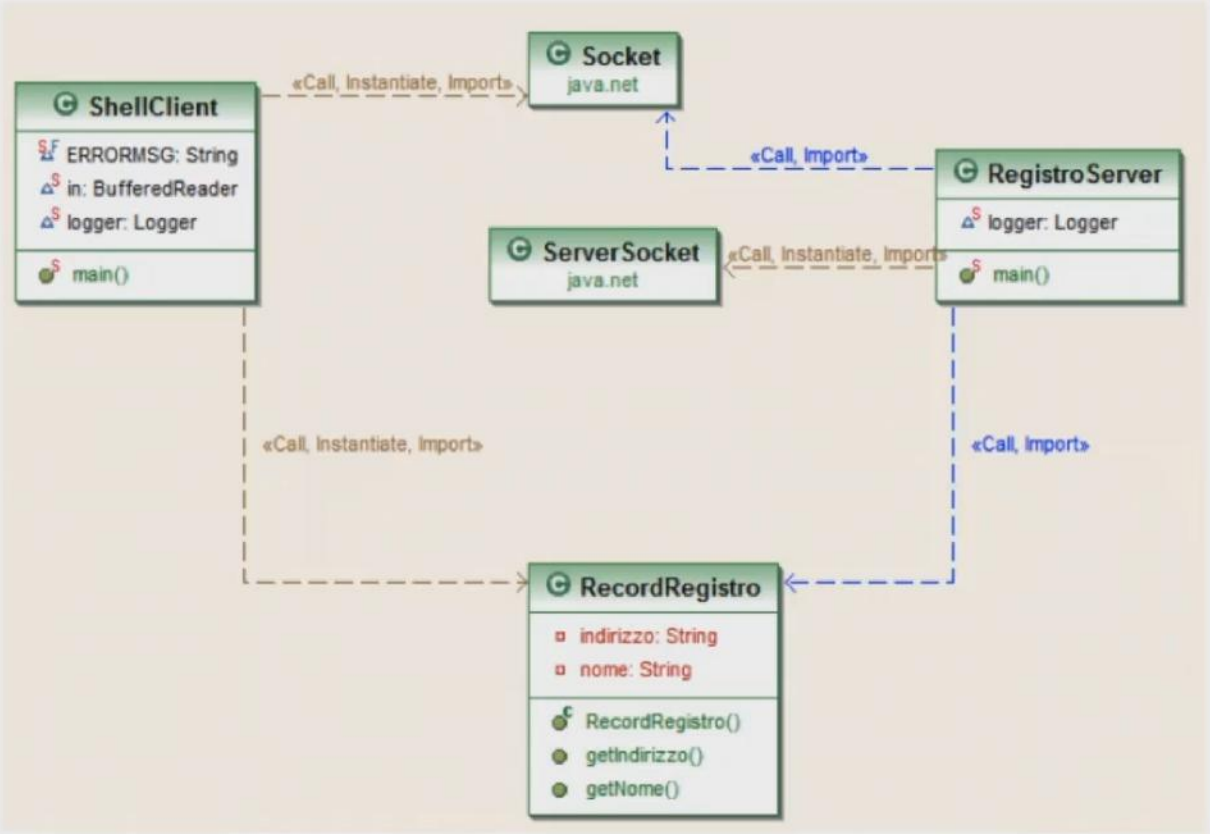
- Import
- Classe Logger
- Mantiene le coppie nome, valore
- Socket di connessione Ciclo
- accetta connessioni
  - crea stream input
  - legge l'oggetto (bloccante)

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.logging.Logger;

public class RegistroServer {
 static Logger logger = Logger.getLogger("global");

 public static void main(String[] args) {
  HashMap<String, RecordRegistro> hash = new HashMap<String,RecordRegistro>();
  Socket socket = null;
  System.out.println ("In attesa...");
  try{
   ServerSocket serverSocket = new ServerSocket(7000);

   while(true) {
    socket = serverSocket.accept();
    ObjectInputStream inStream = new ObjectInputStream(socket.getInputStream());
    RecordRegistro record = (RecordRegistro)inStream.readObject();

    //...
```

RegistroServer (2)

```
    if(record.getIndirizzo()!=null) {
      //scrittura
      hash.put(record.getNome(), record);
    } else {
      //ricerca
      RecordRegistro res=hash.get(record.getNome());
      ObjectOutputStream outStream =
        new ObjectOutputStream(socket.getOutputStream());
      outStream.writeObject(res);
      outStream.flush();
    } //fine else
    socket.close();
  } //fine while
 } catch(EOFException e) {
  logger.severe(e.getMessage());
  e.printStackTrace();
 } catch(Throwable t) {
  logger.severe(t.getMessage());
  t.printStackTrace();
 } finally {
  // chiusura del socket
  try{
    socket.close();
  } catch(IOException e) {
    e.printStackTrace();
    System.exit(0);
  }
 } // end while
} // end main
} // end class
```

La classe `ShellClient`



Come funziona `ShellClient`

```
public class ShellClient {
  public static void main(String args[]) {
    //...
    in = new BufferedReader(newInputStreamReader(System.in));
    try{
       while(!(cmd = ask(">>>")).equals("quit")) {
         if(cmd.equals ("xxxxx")) {
            //...
         } elseif(cmd.equals ("yyyyy")) {
            //...
         } elseif(cmd.equals ("zzzzz")) {
            //...
         } else System.out.println (ERRORMSG);
    } catch(Throwable t) {
       //...
  }

  private static String ask(String prompt) throws  IOException {
   System.out.print(prompt+"");
   return(in.readLine());
  }

  //...
  static final String ERRORMSG ="Cosa?";
  static BufferedReader in = null;
}
```

ShellClient: Il ciclo `while`

- si assegna la stringa alla variabile cmd

```
ask(">>>")
```

- l'operazione di assegnazione assume il valore dell'assegnazione, cioè la stringa digitata dall'utente, che confrontiamo con "quit"

```
(cmd = ask (">>>"))
```

```
(cmd = ask (">>>")).equals("quit")
```

- il risultato di questo test viene negato

```
(!(cmd = ask(">>>")).equals("quit"))
```

- risultato: "fin quando l'utente non digita quit"

```
while(!(cmd = ask(">>>")).equals("quit"))
```

ShellClient

```
import java.io.*;
import java.net.*;
import java.util.logging.Logger;

public class ShellClient {
 static Logger logger = Logger.getLogger("global");
 public static void main(String args[]) {
  String host = args[0];
  String cmd;
  in = new BufferedReader(new InputStreamReader(System.in));
  try{
   while(!(cmd = ask (">>>")).equals("quit")) {
   if(cmd.equals ("inserisci")) {
     System.out.println ("Inserire i dati.");
     String nome = ask("Nome:");
     String indirizzo = ask("Indirizzo:");
     RecordRegistro r = new RecordRegistro(nome, indirizzo);
     Socket socket = new Socket(host, 7000);
     ObjectOutputStream sock_out = new ObjectOutputStream(socket.getOutputStream());
     sock_out.writeObject(r);
     sock_out.flush();
     socket.close();
    } else if(cmd.equals ("cerca")) {
    //...
```

ShellClient

```
 //...
}elseif(cmd.equals ("cerca")) {
 System.out.println ("Inserire il nome per la ricerca.");
 String nome = ask("Nome:");
 RecordRegistro r = new RecordRegistro(nome, null);
 Socket socket = new Socket (host, 7000);
 ObjectOutputStream sock_out = new  ObjectOutputStream(socket.getOutputStream());
 sock_out.writeObject(r);
 sock_out.flush();
 ObjectInputStream sock_in = new ObjectInputStream(socket.getInputStream());
 RecordRegistro result = (RecordRegistro)sock_in.readObject();
 if(result != null)
   System.out.println ("Indirizzo:"+ result.getIndirizzo());
 else
   System.out.println ("Record assente");
 socket.close();
}
  else System.out.println (ERRORMSG);
}//end while
```

ShellClient

- Se si chiede di ricercare
- si crea un record con campo indirizzo vuoto
- si apre un socket
- si preleva lo stream
- si invia l'oggetto
- facendo il flush
- risposta
- Se il risultato è non nullo
  - si stampa
- . . . o non esiste
- Errore comando

```
 //...
}elseif(cmd.equals ("cerca")) {
 System.out.println ("Inserire il nome per la ricerca.");
 String nome = ask("Nome:");
 RecordRegistro r = new RecordRegistro(nome, null);
 Socket socket = new Socket (host, 7000);
 ObjectOutputStream sock_out = new  ObjectOutputStream(socket.getOutputStream());
 sock_out.writeObject(r);
 sock_out.flush();
 ObjectInputStream sock_in = new ObjectInputStream(socket.getInputStream());
 RecordRegistro result = (RecordRegistro)sock_in.readObject();
 if(result != null)
   System.out.println ("Indirizzo:"+ result.getIndirizzo());
 else
   System.out.println ("Record assente");
 socket.close();
}
  else System.out.println (ERRORMSG);
}//end while
```

ShellClient

- Tutte le eccezioni vengono loggate
- Metodo di servizio per input da tastiera
- Legge da stdin (e può lanciare eccezione)
- Stringa di errore

```
  //...

   } catch(Throwable t) {
     logger.severe("Lanciata Throwable:"+ t.getMessage());
     t.printStackTrace();
   }
   System.out.println ("Byebye");
} // fine main

 private static String ask(String prompt) throws IOException {
   System.out.print(prompt+"");
   return(in.readLine());
 }

 static final String ERRORMSG ="Cosa?";

 static BufferedReader in = null;
}
```

ShellClient

- Se si chiede di ricercare
- si crea un record con campo indirizzo vuoto
- si apre un socket
- si preleva lo stream
- si invia l'oggetto
- facendo il flush
- risposta
- Se il risultato è non nullo
  - si stampa
- . . . o non esiste
- Errore comando

```
 //...
}elseif(cmd.equals ("cerca")) {
 System.out.println ("Inserire il nome per la ricerca.");
 String nome = ask("Nome:");
 RecordRegistro r = new RecordRegistro(nome, null);
 Socket socket = new Socket (host, 7000);
 ObjectOutputStream sock_out = new  ObjectOutputStream(socket.getOutputStream());
 sock_out.writeObject(r);
 sock_out.flush();
 ObjectInputStream sock_in = new ObjectInputStream(socket.getInputStream());
 RecordRegistro result = (RecordRegistro)sock_in.readObject();
 if(result != null)
   System.out.println ("Indirizzo:"+ result.getIndirizzo());
 else
   System.out.println ("Record assente");
 socket.close();
}
  else System.out.println (ERRORMSG);
}//end while
```

Conclusioni

- Programmazione con i socket
  - Socket TCP
  - Stream
- Alcuni esempi di uso dei socket
  - "Hello World"
  - Un registro di nomi con architettura client-server
- Conclusioni