



**Corso di Laurea in Informatica**  
**III Anno Triennale**  
**Programmazione Distribuita – classe 1**



# I socket

Delfina Malandrino

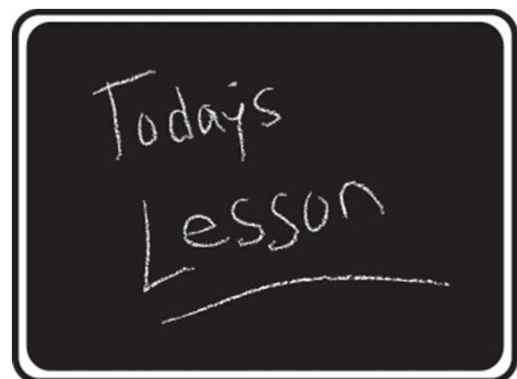
[dmalandrino@unisa.it](mailto:dmalandrino@unisa.it)

<http://www.unisa.it/docenti/delfinamalandrino>

1

## 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



2

1

## Obiettivi della lezione

- Introdurre i socket TCP e la loro programmazione in Java

3

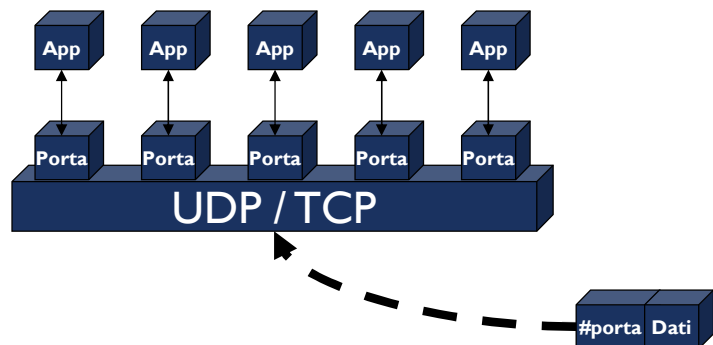
## Organizzazione della lezione

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

4

## Richiami di TCP: porte

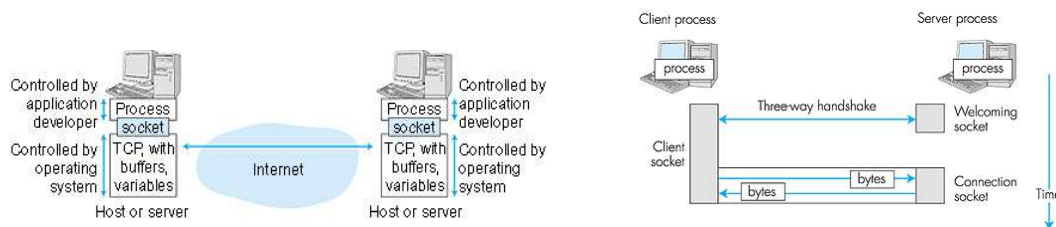
- I protocolli TCP e UDP usano le porte per mappare i dati in ingresso con un particolare processo attivo su un computer
- Ogni socket è legato a un numero di porta così che il livello TCP possa identificare l'applicazione a cui i dati devono essere inviati



5

## 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



6

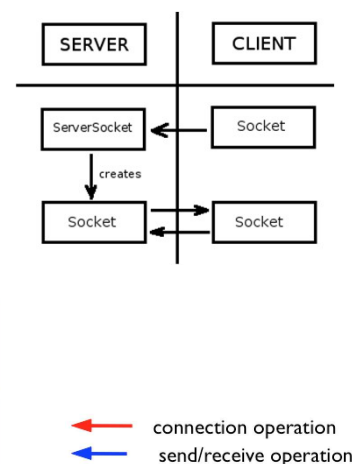
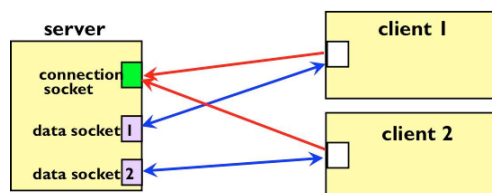
## I socket TCP in Java

- 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

7

## Come funzionano i socket

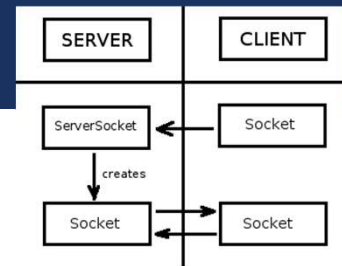
- I socket su stream sono supportati da due classi:
  - **ServerSocket:**
    - per accettare connessioni (socket di connessione)
  - **Socket:**
    - per scambio di dati (socket di dati)



8

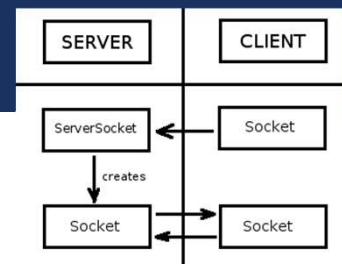
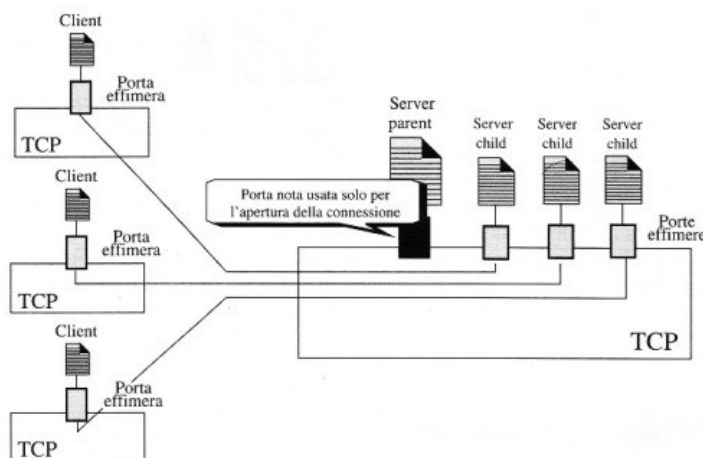
## I metodi di **Socket**

- **ServerSocket(int port)**
  - Crea un server socket su una specifica porta
- **Socket accept() throws IOException**
  - Aspetta connessioni sul ServerSocket e le accetta. Il metodo è bloccante fino a quando non viene fatta una connessione
- **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`



9

## Come funzionano i socket

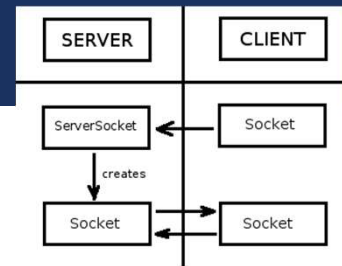


10

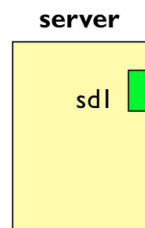
5

## La costruzione di uno stream socket - 1

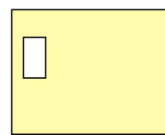
- **sd1** - socket di ascolto



1. Il server stabilisce un server socket sd1 con indirizzo locale per ascoltare connessioni



### client

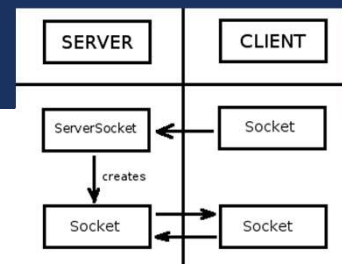


- Il client stabilisce una connessione con il server

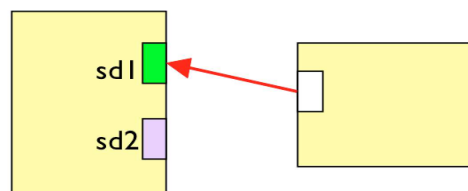
11

## La costruzione di uno stream socket - 2

- **sd2** - socket di connessione
  - o usato per lo scambio dei dati con un client



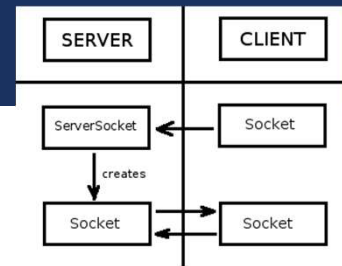
2. Il server accetta una connessione e crea un nuovo socket per dati sd2



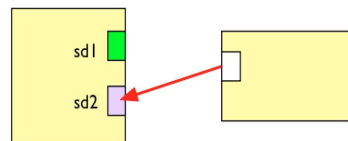
12

## La costruzione di uno stream socket - 3

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



3. Il server effettua una receive

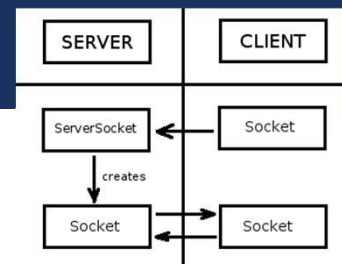


- Il client effettua una send

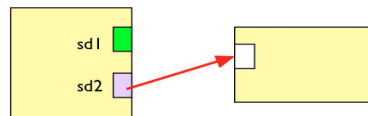
13

## La costruzione di uno stream socket - 4

- Scambio di dati sul socket di connessione sd2
- Il server risponde



4. Il server invia la risposta

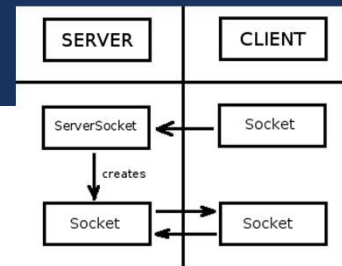


- Il client effettua una receive

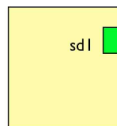
14

## La costruzione di uno stream socket - 5

- Il server chiude la connessione



- Quando il protocollo termina, il server chiude ed elimina il socket sd2



- Il client chiude il socket

15

## 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

16



## Cosa fanno i programmi?

- “Prendi dati, fanne qualcosa, memorizza il risultato”
- Gli stream sono una maniera per prendere dati e trasferirli da qualche altra parte
  - i device sono spesso trattati come stream
- Esempio: un file viene visto come uno stream, un socket viene visto come uno stream, etc.

17

## Cosa è uno stream?

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

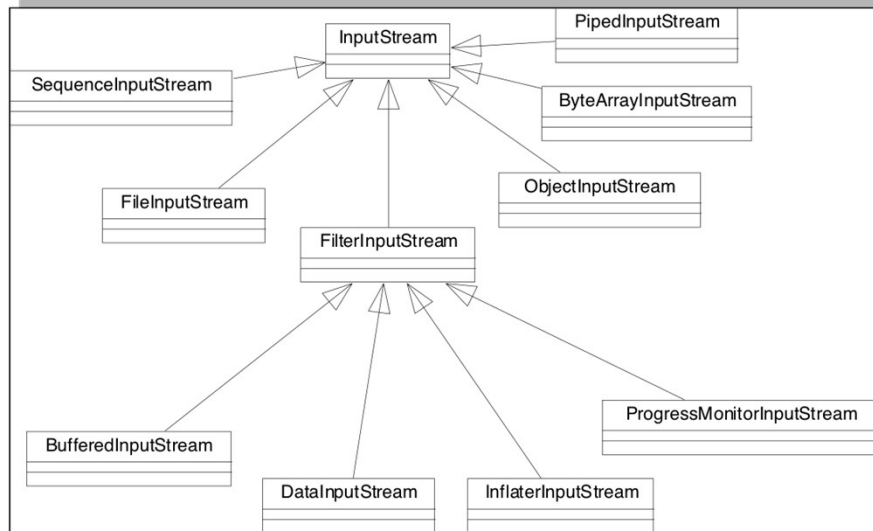
```
int read()
int read(byte[] b)
int read(byte[] b, int off, int len)
void mark(int readlimit)
boolean markSupported()
void reset()
void close()
int available()
long skip(long n)
```

- **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)
```

18

## Cosa è uno stream?



19

## Stream di Input e di Output

- Gli stream sono come i carabinieri della famosa barzelletta: 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



20

10

## Alcuni metodi utili di Stream

- Oggetto della classe `Socket`: il metodo `getInputStream()`
  - restituisce l'input stream associato al socket
  - esiste anche la versione per l'output stream
- Oggetto della classe `ObjectInputStream`: il metodo `readObject()`
  - bloccante
  - restituisce l'oggetto letto dallo stream
  - necessario il casting
- Metodo `writeObject()` di `ObjectOutputStream`

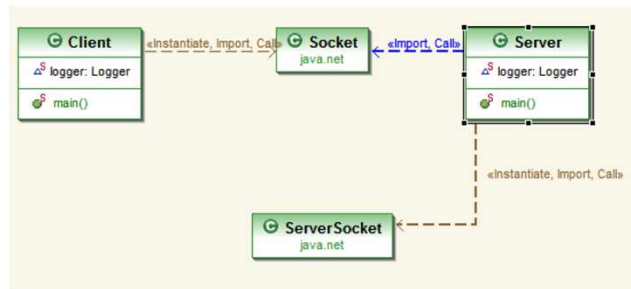
21

## 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

22

## Il diagramma delle classi



23

## Il client

```

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("Delfina");
            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();
        }
    }
}

```

24

## Il server

```
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); ← Dichiarazione del socket di connessione

            logger.info("Socket ok, accetto conn...");

            Socket socket = serverSocket.accept(); ← Accettazione di connessioni

            logger.info("Accettata una connessione..."); ← Quando c'è un client...

            ObjectOutputStream oS = new ObjectOutputStream (socket.getOutputStream()); ←
            ObjectInputStream iS = new ObjectInputStream (socket.getInputStream()); ←
                                                    Si prendono l'output e
                                                    l'input stream
        }
    }
}
```

25

## Il server

```
//...

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();
}
}
```

26

## 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

27

## Struttura

- Un semplice esempio
- Il server mantiene dei record (composti da nome e indirizzo)
- Il client può:



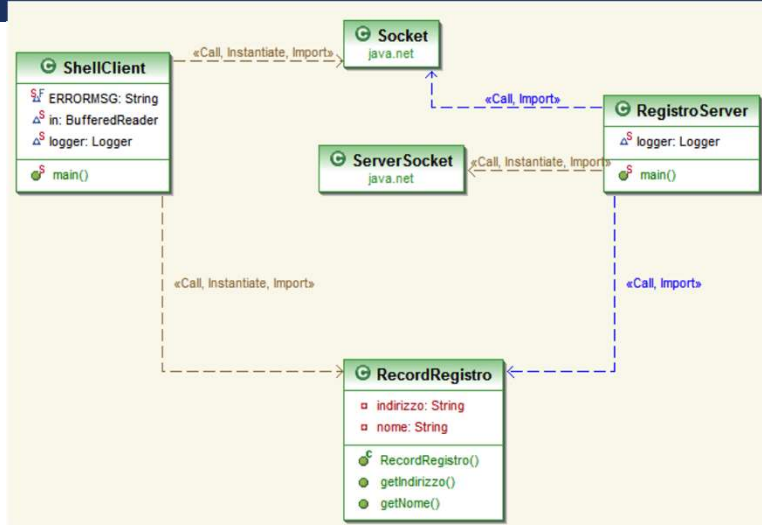
- **inserire un record**: passando un oggetto da inserire
  - record verrà memorizzato



- **ricercare un record**: passando un oggetto solamente con il campo nome inserito
  - ricerca di un record, restituzione di un valore (indirizzo)

28

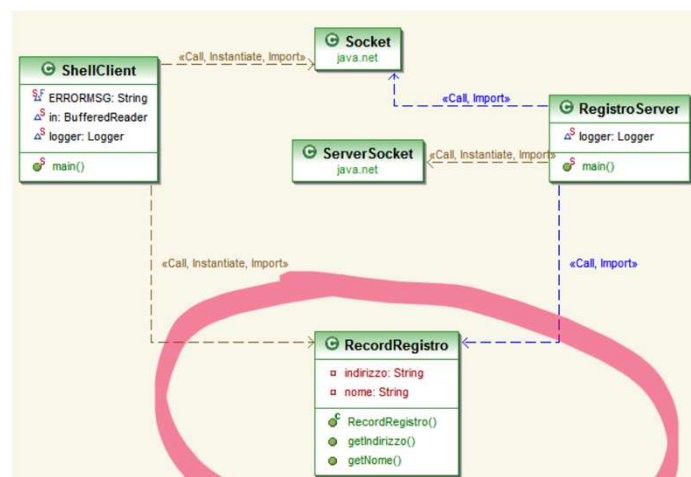
## Diagramma delle classi



29

29

## La classe RecordRegistro



30

15

## La classe RecordRegistro

```
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;
    }

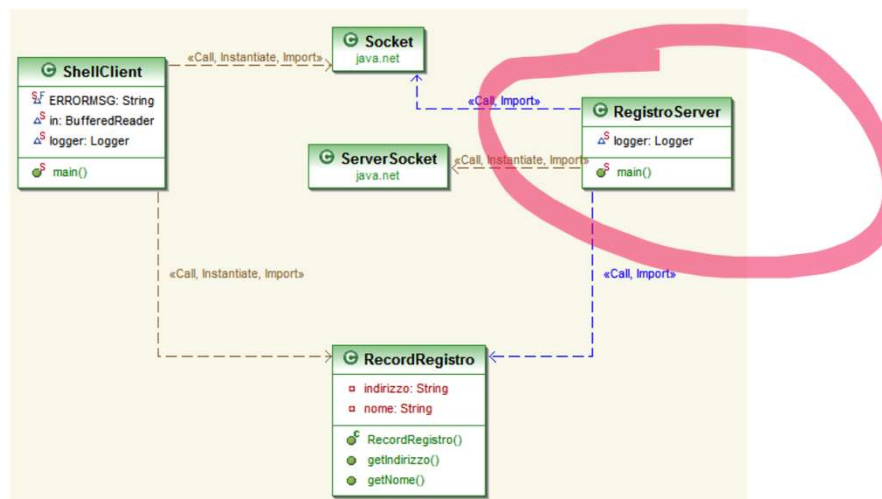
    public String getNome() {
        return nome;
    }

    public String getIndirizzo() {
        return indirizzo;
    }

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

31

## La classe RegistroServer



32

16



## La classe RegistroServer

```
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();
                //...
            }
        } catch (Exception e) {
            logger.severe(e.getMessage());
            e.printStackTrace();
        }
    }
}
```


← Mantiene le coppie nome, valore  
 ← Socket di connessione  
 ← Accetta connessioni  
 ← Crea stream input  
 ← Legge l'oggetto (bloccante)

33

## La classe RegistroServer

```
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);
    }
}
}
```

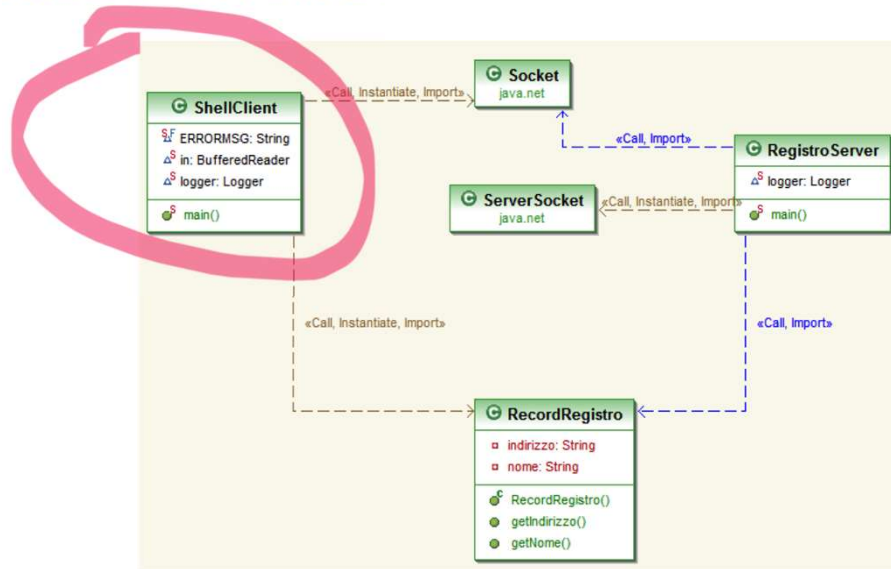
← Record non vuoto: INSERIMENTO  
 ← Record vuoto: RICERCA  
 ← Cerca il NOME  
 ← Crea l'output  
 ← Il record trovato (null se non c'è)  
 ← Chiusura



34

17

## La classe ShellClient



35

## La classe 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();
                }elseif(cmd.equals("cerca")) {
                    //...
                }
            }
        }
    }
}

```

36

## La classe ShellClient

```
//... ricerca

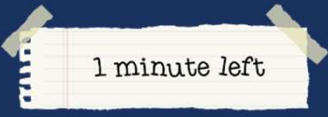
}elseif(cmd.equals ("cerca")) {
    System.out.println ("Inserire il nome per la ricerca.");
    String nome = ask("Nome:");
    RecordRegistro r = new RecordRegistro(nome, null); ← Si crea un record con campo indirizzo vuoto
    Socket socket = new Socket (host, 7000);
    ObjectOutputStream sock_out = 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
//...
} catch(Throwable t) {
    logger.severe("Lanciata Throwable:"+t.getMessage());
    t.printStackTrace();
}
System.out.println ("Bye bye");
} //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;
}
```

37

## Conclusioni


1 minute left

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



### Nelle prossime lezioni:

- ☐ Thread
- ☐ Java Remote Invocation (RMI)

38

19