Programmazione dei socket in Java

Tecnologie e progettazione di sistemi informatici e di telecomunicazioni

Versione n. 13 del 2/12/2025

prof. Roberto FULIGNI

Istituto Tecnico Tecnologico
"Giacomo Fauser"
Novara

Programmazione dei socket in Java

1. Socket TCP e stream di caratteri Unicode

Due applicazioni si scambiano, in modo affidabile, messaggi costituiti da una o più linee di testo terminanti con il carattere *newline*.

Esempio n. 1.1 (Messaggio a singola linea)

Scrivere un’applicazione client/server in cui il client invia al server una frase e il server la restituisce al client convertita in maiuscolo.

Caratteristiche delle applicazioni

- **Server**: supporta l’accesso da parte di più client ma serve un client alla volta (elaborazione sequenziale); elabora una richiesta composta da una sola linea di testo in formato UTF-8 (Unicode); restituisce una risposta formata da una sola linea di testo UTF-8.
- **Client**: Invia una richiesta composta da una sola linea di testo UTF-8 delimitata dal carattere *newline*; elabora una risposta a singola linea di testo UTF-8.

Server (codifica Java)

```
public class ServerMaiuscolo {
    final static int portaServer = 4000;
    public static void main(String[] args) {

        try (ServerSocket server = new ServerSocket(portaServer)) {
            System.out.format("Server in ascolto su: %s%n",
                    server.getLocalSocketAddress());

            // Il server accetta e serve un client alla volta
            while (true) {
                try (Socket client = server.accept()) {
                    String rem = client.getRemoteSocketAddress().toString();
                    System.out.format("Client (remoto): %s%n", rem);

                    comunica(client);

                } catch (IOException e) {
                    System.err.println(String.format("Errore durante la comunicazione
                    con il client: %s", e.getMessage()));
                }
            }
        } catch (IOException e) {
            System.err.println(String.format("Errore server: %s", e.getMessage()));
        }
    }

    private static void comunica(Socket sck) throws IOException {
        BufferedReader in = new BufferedReader(
             new InputStreamReader(sck.getInputStream(), "UTF-8"));
        PrintWriter out = new PrintWriter(
             new OutputStreamWriter(sck.getOutputStream(), "UTF-8"), true);
```

```
        System.out.println("In attesa di ricevere informazioni dal client...");
        String inStr = in.readLine();
        System.out.format("Ricevuto dal client: %s%n", inStr);
        String outStr = elabora(inStr);
        out.println(outStr);
        System.out.format("Inviato al client: %s%n", outStr);
    }

    private static String elabora(String s) {
        return s.toUpperCase();
    }
}
```

Client (codifica Java)

```
public class ClientMaiuscolo {
    final static String nomeServer = "localhost";
    final static int portaServer = 4000;

    public static void main(String[] args) {
        System.out.println("Connessione al server in corso...");
        try (Socket sck = new Socket(nomeServer, portaServer)) {

            String rem = sck.getRemoteSocketAddress().toString();
            String loc = sck.getLocalSocketAddress().toString();
            System.out.format("Server (remoto): %s%n", rem);
            System.out.format("Client (client): %s%n", loc);

            comunica(sck);

        } catch (UnknownHostException e) {
            System.err.format("Nome di server non valido: %s%n", e.getMessage());
        } catch (IOException e) {
            System.err.format("Errore durante la comunicazione con il server: %s%n",
                    e.getMessage());
        }
    }

    private static void comunica(Socket sck) throws IOException {
        BufferedReader in = new BufferedReader(
             new InputStreamReader(sck.getInputStream(), "UTF-8"));
        PrintWriter out = new PrintWriter(
             new OutputStreamWriter(sck.getOutputStream(), "UTF-8"), true);

        Scanner s = new Scanner(System.in, "UTF-8");
        System.out.print("Frase: ");
        String frase = s.nextLine();
        System.out.format("Invio al server: %s%n", frase);
        out.println(frase);
        System.out.println("In attesa di risposta dal server...");
        String risposta = in.readLine();
        System.out.format("Risposta dal server: %s%n", risposta);
    }
}
```

Esempio n. 1.2 (Messaggio multilinea)

Scrivere un’applicazione client/server in cui il client invia al server una sequenza di linee di testo, ciascuna contenente una parola (la sequenza è conclusa con l’invio di una riga vuota). Il server restituisce un messaggio multilinea contenente le sole parole palindrome (una parola per linea, anche il messaggio di risposta termina con una riga vuota).

Caratteristiche delle applicazioni

- **Server**: supporta l’accesso da parte di più client ma serve un client alla volta (elaborazione sequenziale); elabora una richiesta composta da più linee di testo in formato UTF-8 (l’ultima linea è vuota); restituisce una risposta formata da più linee testo UTF-8 (l’ultima linea del messaggio di risposta è vuota).
- **Client**: Invia una richiesta composta da più linee di testo UTF-8 delimitata dal carattere *newline*; elabora una risposta a singola linea di testo UTF-8.

Tutte le linee di testo inviate o ricevute terminano con un carattere di *newline.*

Server (codifica Java)

```
public class ServerPalindrome {
    final static int portaServer = 4000;
    public static void main(String[] args) {

        try (ServerSocket server = new ServerSocket(portaServer)) {
            System.out.format("Server in ascolto su: %s%n",
             server.getLocalSocketAddress());

            // Il server accetta e serve un client alla volta
            while (true) {
                try (Socket client = server.accept()) {
                    String rem = client.getRemoteSocketAddress().toString();
                    System.out.format("Client (remoto): %s%n", rem);
                    comunica(client);
                } catch (IOException e) {
                    System.err.println(String.format("Errore durante la comunicazione
                          con il client: %s", e.getMessage()));
                }
            }
        } catch (IOException e) {
            System.err.println(String.format("Errore server: %s", e.getMessage()));
        }
    }

    private static void comunica(Socket sck) throws IOException {
        BufferedReader in = new BufferedReader(
             new InputStreamReader(sck.getInputStream(), "UTF-8"));
        PrintWriter out = new PrintWriter(
             new OutputStreamWriter(sck.getOutputStream(), "UTF-8"), true);

        ArrayList<String> richiesta = new ArrayList<String>();
        String inStr;
```

```
        System.out.println("In attesa di ricevere informazioni dal client...");
        do {
            inStr = in.readLine().trim();       // Rimuove eventuali spazi superflui
            System.out.format("Ricevuto dal client: %s%n", inStr);
            if (inStr.isEmpty() == false)
                // Si aggiungono solo le stringhe non vuote
                richiesta.add(inStr);
        }
        while (inStr.isEmpty() == false);

        ArrayList<String> risposta = elabora(richiesta);
        for(String outStr : risposta) {
            System.out.format("Inviato al client: %s%n", outStr);
            out.println(outStr);
        }
        out.println();      // Conclude la risposta con una linea vuota
    }

    private static ArrayList<String> elabora(ArrayList<String> richiesta) {
        ArrayList<String> ris = new ArrayList<String>();
        for(String s : richiesta) {
            if (stringaPalindroma(s) == true) {
                ris.add(s);
            }
        }
        return ris;
    }

    private static boolean stringaPalindroma(String s) {
        StringBuilder sb = new StringBuilder(s);

      // Inverte la posizione dei caratteri contenuti in sb e restituisce
      // la nuova sequenza sotto forma di stringa

        String s2 = sb.reverse().toString();

        // Restituisce vero se e solo se le stringhe s e s2 sono uguali
       // (confronto case insensitive).
        return (s.compareToIgnoreCase(s2) == 0);
    }
}
```

Client (codifica Java)

```
public class ClientPalindrome {
    final static String nomeServer = "localhost";
    final static int portaServer = 4000;

    public static void main(String[] args) {
        System.out.println("Connessione al server in corso...");
        try (Socket sck = new Socket(nomeServer, portaServer)) {

            String rem = sck.getRemoteSocketAddress().toString();
            String loc = sck.getLocalSocketAddress().toString();
```

```
            System.out.format("Server (remoto): %s%n", rem);
            System.out.format("Client (client): %s%n", loc);

            comunica(sck);

        } catch (UnknownHostException e) {
            System.err.format("Nome di server non valido: %s%n", e.getMessage());
        } catch (IOException e) {
            System.err.format("Errore durante la comunicazione con il server: %s%n",
                    e.getMessage());
        }
    }

    private static void comunica(Socket sck) throws IOException {
        BufferedReader in = new BufferedReader(
             new InputStreamReader(sck.getInputStream(), "UTF-8"));
        PrintWriter out = new PrintWriter(
             new OutputStreamWriter(sck.getOutputStream(), "UTF-8"), true);

        Scanner s = new Scanner(System.in, "UTF-8");
        String parola;

        do {
            System.out.print("Parola (premere il solo tasto INVIO per terminare): ");
            parola = s.nextLine().trim();
            if (parola.isEmpty() == false) {
                System.out.format("Invio al server: %s%n", parola);
                out.println(parola);
            }
        }
        while (parola.isEmpty() == false);

        out.println(parola);    // Invio di una linea vuota

        System.out.println("Le parole palindrome sono:");
        do {
            parola = in.readLine();
            if (parola.isEmpty() == false) {
                System.out.println(parola);
            }
        }
        while (parola.isEmpty() == false);
    }
}
```

2. Socket TCP e stream di byte

Due applicazioni si scambiano, in modo affidabile, messaggi codificati in binario.

Esempio n. 2.1 (Messaggio a struttura binaria)

Scrivere un’applicazione client/server per elaborare una semplice statistica sui lati di un dato poligono: il client invia il numero intero N di lati del poligono e la lunghezza (di tipo reale) di ciascuno dei lati; il server restituisce due numeri reali, corrispondenti alle lunghezze dei lati più corto e più lungo.

Caratteristiche delle applicazioni

- **Server**: supporta l’accesso da parte di più client ma serve un client alla volta (elaborazione sequenziale); elabora una richiesta in formato binario costituita da un numero di tipo *int* e da N numeri di tipo *double*; restituisce una statistica composta da due dati di tipo *double* (max e min).
- **Client**: Invia una richiesta in formato binario secondo le specifiche indicate e riceve la statistica prodotta dal server.

Server (codifica Java)

```
public class ServerPoligono {
    final static int portaServer = 4500;
    public static void main(String[] args) {

        try (ServerSocket server = new ServerSocket(portaServer)) {
            System.out.format("Server in ascolto su: %s%n",
                    server.getLocalSocketAddress());

            // Il server accetta e serve un client alla volta
            while (true) {
                try (Socket client = server.accept()) {
                    String rem = client.getRemoteSocketAddress().toString();
                    System.out.format("Client (remoto): %s%n", rem);
                    comunica(client);
                } catch (IOException e) {
                    System.err.println(String.format("Errore durante la comunicazione
                            con il client: %s", e.getMessage()));
                }
            }
        } catch (IOException e) {
            System.err.println(String.format("Errore server: %s", e.getMessage()));
        }
    }

    private static void comunica(Socket sck) throws IOException {
        DataInputStream in = new DataInputStream(
             new BufferedInputStream(sck.getInputStream()));
        DataOutputStream out = new DataOutputStream(
             new BufferedOutputStream(sck.getOutputStream()));

        int numLati;
        ArrayList<Double> lati = new ArrayList<Double>();
```

```
        System.out.println("In attesa di ricevere informazioni dal client...");
        numLati = in.readInt();
        System.out.format("Ricevo dal client un poligono di %d lati:%n", numLati);
        for (int i = 0; i < numLati; i++) {
            Double lato = in.readDouble();
            lati.add(lato);
            System.out.format(" %.2f", lato);
        }
        System.out.println();
        ArrayList<Double> statistica = elabora(lati);
        double min = statistica.get(0);
        double max = statistica.get(1);
        out.writeDouble(min);
        out.writeDouble(max);
        out.flush();    // Svuota il buffer
        System.out.format("Inviata al client la statistica (Min: %.2f   Max: %.2f)%n",
             min, max);
    }

    private static ArrayList<Double> elabora(ArrayList<Double> lati) {
        ArrayList<Double> stat = new ArrayList<Double>();
        stat.add(Collections.min(lati));
        stat.add(Collections.max(lati));
        return stat;
    }
}
```

Client (codifica Java)

```
public class ClientPoligono {
    final static String nomeServer = "localhost";
    final static int portaServer = 4500;

    public static void main(String[] args) {
        System.out.println("Connessione al server in corso...");
        try (Socket sck = new Socket(nomeServer, portaServer)) {

            String rem = sck.getRemoteSocketAddress().toString();
            String loc = sck.getLocalSocketAddress().toString();
            System.out.format("Server (remoto): %s%n", rem);
            System.out.format("Client (client): %s%n", loc);

            comunica(sck);

        } catch (UnknownHostException e) {
            System.err.format("Nome di server non valido: %s%n", e.getMessage());
        } catch (IOException e) {
            System.err.format("Errore durante la comunicazione con il server: %s%n",
                    e.getMessage());
        }
    }

    private static void comunica(Socket sck) throws IOException {
        DataInputStream in = new DataInputStream(
             new BufferedInputStream(sck.getInputStream()));
        DataOutputStream out = new DataOutputStream(
```

```
                new BufferedOutputStream(sck.getOutputStream()));

        Scanner s = new Scanner(System.in);
        int numLati;
        do {
            System.out.print("Numero di lati del poligono: ");
            numLati = s.nextInt();
        }
        while (numLati < 3);
        out.writeInt(numLati);
        for (int i = 1; i <= numLati; i++) {
            double lato;
            do {
                System.out.format("lunghezza del lato n. %d: ", i);
                lato = s.nextDouble();
            }
            while (lato <= 0);
            out.writeDouble(lato);
        }
        out.flush();

        double min = in.readDouble();
        double max = in.readDouble();
        System.out.format("Statistica ricevuta dal server: min = %.2f  max: %.2f%n",
                min, max);
    }
}
```

Esempio n. 2.2 (Messaggi composti da oggetti serializzabili)

Scrivere un’applicazione client/server per eseguire una ricerca nel database degli impiegati di un’azienda: il client invia un oggetto contenente il cognome e il nome dell’impiegato; il server restituisce un altro oggetto in cui sono inseriti i risultati della ricerca (cognome, nome, matricola, stipendio).

Caratteristiche delle applicazioni

- Il client e il server condividono la definizione di due classi *Ricerca* e *Risultato.* Queste classi definiscono la struttura dei messaggi scambiati e devono implementare l’interfaccia *Serializable*
- **Server**: supporta l’accesso da parte di più client ma serve un client alla volta (elaborazione sequenziale); riceve dal client un oggetto di tipo *Ricerca* da cui ricava i criteri di ricerca; genera e restituisce al client un oggetto di tipo *Risultato* contenente i dati dell’impiegato trovato (oppure *null* se l’impiegato non esiste).
- **Client**: legge da tastiera il cognome e il nome di un impiegato, inserisce i dati in un oggetto di tipo *Ricerca* che invia al server; riceve un oggetto di tipo *Risultato* da cui ricava il risultato della ricerca.

Server (codifica Java)

```
class Ricerca implements Serializable {
    public String cognome;
    public String nome;

    public Ricerca(String cognome, String nome) {
        this.cognome = cognome;
        this.nome = nome;
    }
}

class Risultato implements Serializable {
    public String cognome;
    public String nome;
    public int matricola;
    public double stipendio;

    public Risultato(String cognome, String nome, int matricola, double stipendio) {
        this.cognome = cognome;
        this.nome = nome;
        this.matricola = matricola;
        this.stipendio = stipendio;
    }
}

public class ServerImpiegato {
    final static int portaServer = 5000;
    public static void main(String[] args) {

        try (ServerSocket server = new ServerSocket(portaServer)) {
            System.out.format("Server in ascolto su: %s%n",
                    server.getLocalSocketAddress());

            // Il server accetta e serve un client alla volta
            while (true) {
                try (Socket client = server.accept()) {
                    String rem = client.getRemoteSocketAddress().toString();
                    System.out.format("Client (remoto): %s%n", rem);
                    comunica(client);
                } catch (IOException e) {
                    System.err.println(String.format("Errore durante la comunicazione
                            con il client: %s", e.getMessage()));
                }
            }
        } catch (IOException e) {
            System.err.println(String.format("Errore server: %s", e.getMessage()));
        }
    }

    private static void comunica(Socket sck) throws IOException {
        ObjectInputStream in = new ObjectInputStream(sck.getInputStream());

        System.out.println("In attesa di ricevere informazioni dal client...");
        Ricerca ric = null;
```

```
        try {
            // Deserializzazione: ricevo una sequenza di byte e
            // la trasformo in un oggetto
            ric = (Ricerca) in.readObject();
        } catch (ClassNotFoundException e) {
            throw new IOException("Tipo di ricerca non supportata.");
        }

        System.out.format("Ricevuta dal client la  ricerca: cognome = '%s'  nome =
             '%s'%n", ric.cognome, ric.nome);

        Risultato ris = elabora(ric);

        ObjectOutputStream out = new ObjectOutputStream(sck.getOutputStream());
        out.writeObject(ris);
        System.out.println("Inviato al client il risultato della ricerca");
    }

    private static Risultato elabora(Ricerca r) {
        Risultato ris;
        boolean trovato;

        trovato = r.cognome.toUpperCase().equals("ROSSI") &&
                  r.nome.toUpperCase().equals("MARIO");

        if (trovato == true) {
            ris = new Risultato("Rossi", "Mario", 10012, 1598.50);
        }
        else {
            ris = null;
        }

        return ris;
    }
```

Client (codifica Java)

```
class Ricerca implements Serializable {
    public String cognome;
    public String nome;

    public Ricerca(String cognome, String nome) {
        this.cognome = cognome;
        this.nome = nome;
    }
}

class Risultato implements Serializable {
    public String cognome;
    public String nome;
    public int matricola;
    public double stipendio;

    public Risultato(String cognome, String nome, int matricola, double stipendio) {
        this.cognome = cognome;
        this.nome = nome;
```

```
        this.matricola = matricola;
        this.stipendio = stipendio;
    }
}

public class ClientImpiegato {
    final static String nomeServer = "localhost";
    final static int portaServer = 5000;

    public static void main(String[] args) {
        System.out.println("Connessione al server in corso...");
        try (Socket sck = new Socket(nomeServer, portaServer)) {

            String rem = sck.getRemoteSocketAddress().toString();
            String loc = sck.getLocalSocketAddress().toString();
            System.out.format("Server (remoto): %s%n", rem);
            System.out.format("Client (client): %s%n", loc);

            comunica(sck);

        } catch (UnknownHostException e) {
            System.err.format("Nome di server non valido: %s%n", e.getMessage());
        } catch (IOException e) {
            System.err.format("Errore durante la comunicazione con il server: %s%n",
                    e.getMessage());
        }
    }

    private static void comunica(Socket sck) throws IOException {
        ObjectOutputStream out = new ObjectOutputStream(sck.getOutputStream());

        Scanner s = new Scanner(System.in);
        System.out.println("Inserire l'impiegato da ricercare.");
        System.out.print("Cognome: ");
        String co = s.nextLine();
        System.out.print("Nome: ");
        String no = s.nextLine();

        Ricerca ric = new Ricerca(co, no);
        out.writeObject(ric);

        System.out.println("In attesa di risposta dal server...");
        Risultato ris = null;

        ObjectInputStream in = new ObjectInputStream(sck.getInputStream());
        try {
            // Deserializzazione: ricevo una sequenza di byte e
            // la trasformo in un oggetto
            ris = (Risultato) in.readObject();
        } catch (ClassNotFoundException e) {
            throw new IOException("Tipo di risultato non supportato.");
        }
        System.out.println("Risultato della ricerca:");
        if (ris != null) {
            System.out.format("Cognome: %s%n", ris.cognome);
            System.out.format("Nome: %s%n", ris.nome);
```

```
            System.out.format("Matricola: %d%n", ris.matricola);
            System.out.format("Stipendio: %.2f Euro%n", ris.stipendio);
        }
        else {
            System.out.println("Impiegato non presente in archivio");
        }
    }
}
```

3. Multiserver TCP

Il server elabora più richieste contemporaneamente distribuendo il carico di lavoro su più *thread* di elaborazione.

Esempio n. 3.1 (Server multithreaded)

Scrivere un’applicazione C/S in cui: il client richiede N numeri interi compresi tra A e B (estremi inclusi); il server genera i numeri richiesti e li restituisce, uno alla volta, al client. Si assuma che il server impieghi due secondi per generare un numero casuale.

Assunzioni: $0 \le N \le 100$; $0 \le A \le B \le 10^6$

Caratteristiche delle applicazioni

- **Server**: supporta l’accesso da parte di più client e **serve quattro client alla volta** (elaborazione parallela multithreaded); elabora una richiesta composta da una sola linea di testo in formato UTF-8 (Unicode) contenente, nell’ordine, i numeri N, A, B; restituisce N linee di testo contenenti i numeri casuali (un numero per linea).
- **Client**: richiede l’inserimento di N, A, B da tastiera; genera e invia al server una linea di testo contenente i tre dati; riceve e stampa a video i numeri casuali.

Server (codifica Java)

```
public class ServerCasualiMT {
    static final int PORTA_SERVER = 5000;
    static final int MAX_THREAD = 4;
    static final int RITARDO = 2000;    // ms

    // Esecutore di task. Utilizza un pool di MAX_THREAD thread
    static ExecutorService esecutore = Executors.newFixedThreadPool(MAX_THREAD);

    // Risorse condivise tra i thread di elaborazione
    static Random rnd = new Random(2019);   // Generatore di numeri casuali

    static Lock lc = new ReentrantLock();    // Usato nell'accesso alle risorse in mutua esclusione

    public static void main(String[] args) {
        try (ServerSocket server = new ServerSocket(PORTA_SERVER)) {
            System.out.format("Server in ascolto su: %s%n", server.getLocalSocketAddress());

            while (true) {
                Socket tempSck;
                try {
                    tempSck = server.accept();
                    esecutore.execute(() -> {
                        // Copia il socket creato dal main thread: in questo modo la variabile
                        // tmpSck può essere utilizzata per accettare nuove connessioni.
                        Socket client = tempSck;
                        try (client) {
                            String rem = client.getRemoteSocketAddress().toString();
                            Thread t = Thread.currentThread();
                            System.out.format("Thread %d - Client (remoto): %s%n", t.getId(), rem);

                            // Avvia la comunicazione con il client. Il socket è automaticamente
```

```
                        // chiuso alla fine di questo blocco TRY/CATCH
                        comunica(client);

                    } catch (IOException e) {
                        System.err.println(String.format("Errore durante la comunicazione con
                                il client: %s", e.getMessage()));
                    }
                });
            } catch (IOException e) {
                System.err.println(String.format("Errore nella gestione di nuove connessioni:
                        %s", e.getMessage()));
            }
        }
    } catch (IOException e) {
        System.err.println(String.format("Errore del server: %s", e.getMessage()));
    }
}

private static void comunica(Socket sck) {
    try (
            BufferedReader in = new BufferedReader(
                  new InputStreamReader(sck.getInputStream(), "UTF-8"));
            PrintWriter out = new PrintWriter(
                  new OutputStreamWriter(sck.getOutputStream(), "UTF-8"), true);
    ) {
        int n, a, b;
        Thread t = Thread.currentThread();

        String str = in.readLine();
        Scanner s = new Scanner(str);
        n = s.nextInt();
        a = s.nextInt();
        b = s.nextInt();

        for(int i = 0; i < n; i++) {
            int r;

            // Accesso in mutua esclusione alla sezione critica
            lc.lock();
            try {
                // Inizio sezione critica
                r = a + rnd.nextInt(b - a + 1);
                // Fine sezione critica
            } finally {
                lc.unlock();
            }

            Thread.sleep(RITARDO);
            System.out.format("\tThread %d - Inviato numero %d%n", t.getId(), r);

            out.println(r);

            // La classe PrintWriter non genera eccezioni. In presenza di un errore
            // di comunicazione occorre lanciare esplicitamente un'eccezione.
            if (out.checkError() == true) {
                throw new IOException("Errore durante la scrittura dei dati.");
            }
        }
    } catch (UnsupportedEncodingException e) {
        System.err.println(String.format("Codifica di caratteri non supportata: %s",
                e.getMessage()));
    } catch (IOException e) {
```

```
            System.err.println(String.format("Errore di I/O: %s", e.getMessage()));
        } catch (InterruptedException e) {  // Richiesto da Thread.sleep
            System.err.println(e.getMessage());
        }
    }
}
```

Client (codifica Java)

```
public class Main {
    final static String nomeServer = "localhost";
    final static int portaServer = 5000;
    final static int MIN_N = 0;
    final static int MAX_N = 100;
    final static int MIN_INTERVALLO = 0;
    final static int MAX_INTERVALLO = 1000000;

    public static void main(String[] args) {
        System.out.println("Connessione al server in corso...");
        try (Socket sck = new Socket(nomeServer, portaServer)) {

            String rem = sck.getRemoteSocketAddress().toString();
            String loc = sck.getLocalSocketAddress().toString();
            System.out.format("Server: %s%n", rem);
            System.out.format("Client: %s%n", loc);

            comunica(sck);

        } catch (UnknownHostException e) {
            System.err.format("Nome di server non valido: %s%n", e.getMessage());
        } catch (IOException e) {
            System.err.format("Errore durante la comunicazione con il server: %s%n",
              e.getMessage());
        }
    }

    private static void comunica(Socket sck) throws IOException {
        try (
            BufferedReader in = new BufferedReader(
                      new InputStreamReader(sck.getInputStream(), "UTF-8"));
            PrintWriter out = new PrintWriter(
                      new OutputStreamWriter(sck.getOutputStream(), "UTF-8"), true);
            Scanner s = new Scanner(System.in, "UTF-8");
        ) {
            int n, a, b;
            do {
                System.out.print("Numero di valori casuali da richiedere: ");
                n = s.nextInt();
            }
            while (n < MIN_N || n > MAX_N);

            do {
                System.out.print("Estremi (inclusi) dell'intervallo: ");
                a = s.nextInt();
                b = s.nextInt();
            }
            while (a < MIN_INTERVALLO || a > b || b > MAX_INTERVALLO);

            System.out.println("Invio della stringa di richiesta...");
            out.println(String.format("%d %d %d", n, a, b));

            System.out.format("I %d numeri casuali ricevuti dal server sono:%n", n);
```

```
            for(int i = 0; i < n; i++) {
                int numero = Integer.parseInt(in.readLine());
                System.out.println(numero);
            }
        }
    }
}
```

4. Socket UDP

Due applicazioni si scambiano, in modo rapido ma non necessariamente affidabile, dati codificati in binario.

Esempio n. 4.1 (Datagramma binario)

Scrivere un’applicazione C/S in cui il client invia un nome di persona e il server risponde con una frase di saluto.

Caratteristiche delle applicazioni

- **Server**: riceve dal client un datagramma UDP contenente i dati della codifica UTF-8 di una stringa; trasforma la sequenza in una stringa da cui ricava il nome della persona; costruisce la frase di saluto, la converte in una sequenza di byte e la inserisce in un nuovo datagramma; invia il datagramma prodotto al client.
- **Client**: richiede l’inserimento da tastiera di un nome (stringa UTF-8); genera e invia al server un datagramma contenente il nome; riceve dal server un nuovo datagramma da cui estrae la frase di saluto; stampa la frase.

Server (codifica Java)

```
public class ServerSaluti {
    final static int portaServer = 6789;
    public static void main(String[] args) {
        byte[] bufferIn = new byte[1024];
        byte[] bufferOut;

        try (DatagramSocket sck = new DatagramSocket(portaServer)) {
            System.out.format("Server in ascolto su %s.", sck.getLocalSocketAddress());
            while(true) {
                DatagramPacket pktIn = new DatagramPacket(bufferIn, bufferIn.length);
                sck.receive(pktIn);

                SocketAddress saClient = pktIn.getSocketAddress();
                String nome = new String(pktIn.getData(), 0, pktIn.getLength(), "UTF-8");

                nome = nome.trim().toUpperCase();
                System.out.format("Ricevuto dal client %s il nome %s... ", saClient, nome);

                String saluto = String.format("Salve %s, ti auguro una buona giornata.%n", nome);
                bufferOut = saluto.getBytes("UTF-8");

                DatagramPacket pktOut = new DatagramPacket(bufferOut, bufferOut.length, saClient);
                sck.send(pktOut);
                System.out.println("Inviato il saluto.");
            }
        } catch (SocketException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client (codifica Java)

```
public class ClientSaluti {
    final static String nomeServer = "localhost";
    final static int portaServer = 6789;
    public static void main(String[] args) {
        byte[] bufferIn = new byte[1024];
        byte[] bufferOut;

        Scanner sc = new Scanner(System.in, "UTF-8");
        System.out.print("Qual è il tuo nome? ");
        String nome = sc.nextLine() + System.lineSeparator();

        try (DatagramSocket sck = new DatagramSocket()) {

            bufferOut = nome.getBytes("UTF-8");
            InetAddress ipServer = InetAddress.getByName(nomeServer);
            DatagramPacket pktOut = new DatagramPacket(bufferOut, bufferOut.length, ipServer, portaServer);
            System.out.format("Invio del nome al server...%n");
            sck.send(pktOut);

            DatagramPacket pktIn = new DatagramPacket(bufferIn, bufferIn.length);
            sck.receive(pktIn);
            String risposta = new String(pktIn.getData(), 0, pktIn.getLength(), "UTF-8");
            risposta = risposta.trim();
            System.out.format("Risposta dal server: %s", risposta);

        } catch (SocketException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Esempio n. 4.2 (Multicast)

Scrivere un’applicazione C/S in cui il server invia periodicamente un aforisma a un gruppo di client.

Caratteristiche delle applicazioni

- **Server**: ripete continuamente la seguente attività: attende per un certo tempo, quindi ricava una stringa di testo contenente un aforisma e la invia a un indirizzo di multicast.
- **Client**: dopo essersi unito al gruppo multicast previsto, riceve e visualizza sei aforismi. Al termine dell’attività il client abbandona il gruppo.

Server (codifica Java)

```
public class ServerAforismi {
    final static String gruppo = "225.6.7.8";
    final static int portaMulticast = 6789;
    final static int ritardo = 8000;        // ms
    final static String[] aforismi = {
            "L'istruzione è l'arma più potente che noi possiamo usare per cambiare il mondo
                (Nelson Mandela)",
            "Tutto quello che puoi fare, o sognare di poter fare, incomincialo. Il coraggio ha in
                sé genio, potere e magia. Incomincia adesso. (Goethe)",
            "Il mondo è un bel posto, per il quale vale la pena di lottare. (Ernest Hemingway)",
            "La ragione e il torto non si dividono mai con un taglio così netto che ogni parte
                abbia soltanto dell'uno e dell'altra... (Alessandro Manzoni)",
            "Due cose sono infinite: l'universo e la stupidità umana, ma riguardo l'universo ho
                ancora dei dubbi. (Albert Einstein)",
            "La natura è il miglior modo per comprendere l'arte; i pittori ci insegnano a
                vedere... (Vincent Van Gogh)"
    };
    public static void main(String[] args) {
        byte[] bufferOut;
        int indice = 0;
        try (MulticastSocket sck = new MulticastSocket()) {
            System.out.println("Server avviato.");
            InetAddress ipMulticast = InetAddress.getByName(gruppo);
            while(true) {
                Thread.sleep(ritardo);
                String af = aforismi[indice] + System.lineSeparator();
                bufferOut = af.getBytes("UTF-8");
                DatagramPacket pktOut = new DatagramPacket(bufferOut, bufferOut.length,
                    ipMulticast, portaMulticast);
                sck.send(pktOut);
                System.out.format("Inviato l'aforisma n. %d all'indirizzo di multicast %s%n",
                    indice, ipMulticast);
                indice = (indice + 1) % aforismi.length;
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Client (codifica Java)

```java
public class ClientAforismi {
    final static String gruppo = "225.6.7.8";
    final static int portaMulticast = 6789;
    final static int maxAforismi = 6;

    public static void main(String[] args) {
        try (MulticastSocket sck = new MulticastSocket(portaMulticast)) {
            System.out.println("Client avviato.");
            InetAddress ipMulticast = InetAddress.getByName(gruppo);
            sck.joinGroup(ipMulticast);
            try {
                for (int i = 1; i <= maxAforismi; i++) {
                    byte[] bufferIn = new byte[1024];
                    DatagramPacket pktIn = new DatagramPacket(bufferIn, bufferIn.length);
                    sck.receive(pktIn);
                    System.out.format("Aforisma n. %d/%d: ", i, maxAforismi);
                    String af = new String(pktIn.getData(), 0, pktIn.getLength(), "UTF-8");
                    System.out.print(af);
                }
            } finally {
                sck.leaveGroup(ipMulticast);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Esercizi risolti

Sistema di estrazioni del lotto

Libro di testo, pag. 135 n.5.

Realizza un sistema di estrazioni del lotto: N giocatori scelgano tre numeri e li comunichino alla ricevitoria. Quindi effettua le estrazioni del lotto comunicando eventuali vincite.

Caratteristiche del server

Server multithread Java composto da:

- *N thread di gestione dei giocatori (worker)*: ognuno di questi thread riceve, valida e memorizza i tre numeri dal giocatore associato, quindi rimane in attesa dell’estrazione; successivamente, confronta i numeri ricevuti con quelli estratti e comunica al giocatore l’esito dell’estrazione.
- *thread di estrazione*: attende che N giocatori abbiano completato l’invio dei numeri, quindi chiude la ricevitoria (chiudendo il server socket in modo tale da non accettare ulteriori connessioni), procede all’estrazione di cinque numeri vincenti distinti e sblocca gli N thread di gestione dei giocatori.
- *main thread*: avvia il thread di estrazione, quindi crea un server socket per accettare connessioni TCP sulla porta 5060; per ogni connessione accettata avvia la routine di gestione del giocatore su un thread distinto; alla chiusura del server socket (effettuata dal thread di estrazione), rimane in attesa del completamento delle attività svolte dagli altri thread

Per sincronizzare i thread dell’applicazione server, si utilizzano i seguenti oggetti:

- ***CountDownLatch giocatoriAttesi***: inizializzato a N, è l’oggetto su cui rimane inizialmente in attesa il thread di estrazione. Ciascun thread worker decrementa questo CountDownLatch dopo aver ricevuto e validato i tre numeri dal giocatore.
- ***CountDownLatch estrazioniAttese***: inizializzato a 1, è l’oggetto su cui rimangono in attesa i worker dopo aver memorizzato i tre numeri. Questo CountDownLatch è decrementato dal thread di estrazione dopo aver generato i numeri vincenti. Il decremento di *estrazioniAttese* sblocca i thread worker, i quali posso completare la loro attività trasmettendo ai giocatori associati l’esito dell’estrazione.

Caratteristiche del client

Applicazione console Java, svolge le seguenti attività:

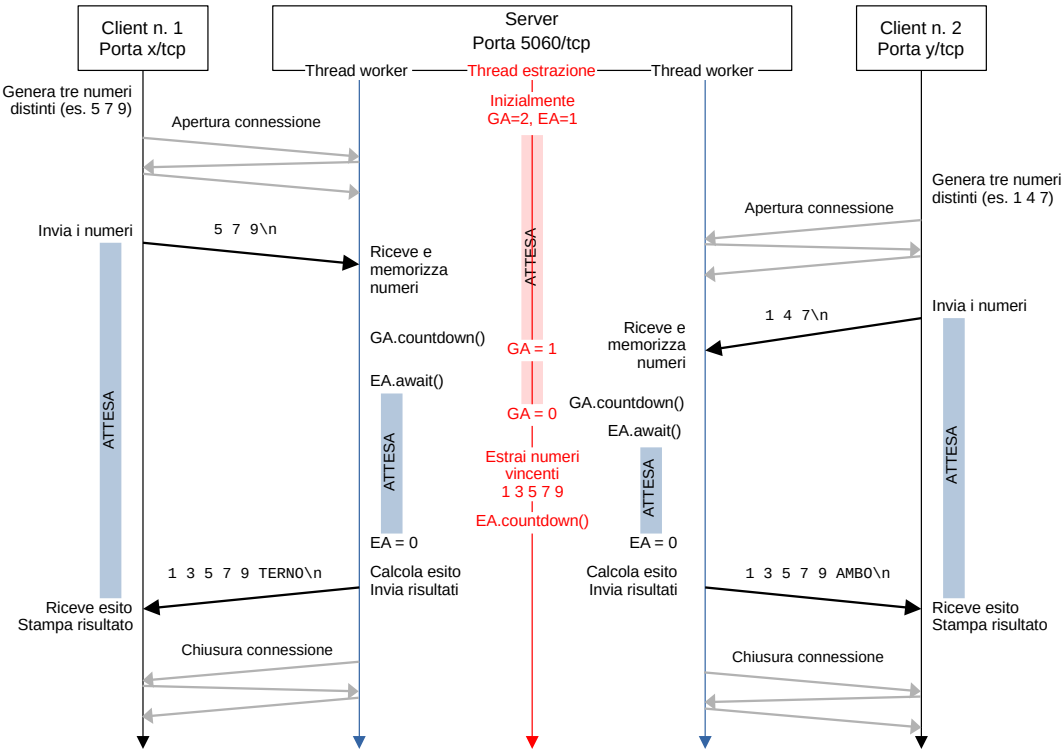
- genera casualmente tre numeri distinti, scelti in un dato intervallo;
- apre una connessione TCP sulla porta 5060 del server;
- invia al server i tre numeri, quindi rimane in attesa dell’esito dell’estrazione
- riceve un messaggio contenente i risultati dell’estrazione, oppure un opportuno messaggio in caso di errore
- visualizza il messaggio ricevuto sullo schermo

Protocollo di comunicazione

Si opta per un protocollo *ad hoc* in base al quale client e server si scambiano messaggi testuali UTF-8 a singola linea.

- *Messaggio contenente i numeri del giocatore*: linea di testo inviata dal client, contiene i tre numeri separati da uno spazio seguiti dal carattere *newline*. *E*sempio: **`5 7 9\n`**
- *Messaggio contenente i numeri del giocatore*: linea di testo inviata dal server, composta dai cinque numeri vincenti (separati da uno spazio), un ulteriore spazio e una frase che riporta l’esito dell’estrazione. I possibili esiti sono: NESSUN NUMERO ESTRATTO, UN NUMERO ESTRATTO, AMBO, TERNO

Lo scambio di messaggi tra N = 2 client e il server e la sincronizzazione dei thread del server sono descritti dal seguente diagramma temporale.



Soluzione

Sulla piattaforma Netlab è disponibile la soluzione dell’esercizio Sistema di estrazioni del lotto. La soluzione consiste in un progetto multimodulo realizzato con il software IntelliJ Idea.

Gestione del tabellone del lotto (multicast)

Libro di testo, pag. 152 (esercizio proposto).

Realizza un sistema per la gestione delle estrazioni del lotto:

- *il server gestisce il tabellone ed effettua una nuova estrazione delle 11 ruote ogni giorno;*
- *l’estrazione su una nuova ruota, tra le 11 definite, avviene ogni 2 minuti, ed è composta da cinque numeri differenti in un range 1÷90, e ha la seguente struttura:*
  `<nome> <estratto1>,<estratto2>,<estratto3>,<estratto4>,<estratto5>`

I singoli numeri vengono inviati man mano che vengono estratti ai diversi client: se un utente si collega mentre si è nel mezzo di un’estrazione, viene messo in attesa dell’inizio di un’estrazione su di una nuova ruota.

A fine giornata il server memorizza i numeri estratti in un file, che ha per nome il numero dell’estrazione e la data (`est20_15_02_2020.txt`*).*

Considerazioni iniziali

Per verificare la correttezza dell’algoritmo di gestione dei tempi di attesa iniziali, si estende la durata dell’estrazione su una singola ruota inserendo un breve ritardo prima dell’estrazione di un nuovo numero.

Poiché è difficile controllare i tempi di attesa di ogni singolo client con una sola trasmissione multicast, si opta per la seguente soluzione:

1. Il server rimane in ascolto sulla porta 5070/tcp (connessioni unicast) e contemporaneamente avvia, mediante un thread addizionale, l’estrazione della prima ruota trasmettendo i numeri risultanti in multicast sulla porta 5070/udp
2. Il client apre una nuova connessione TCP sulla porta 5070 e rimane in attesa di ricevere un opportuno messaggio di inizio estrazione
3. Il server accetta la connessione e inserisce il socket del client in una opportuna lista dei socket in stato d’attesa. Tutti i client che si connettono al server in questa fase sono inseriti nella lista.
4. Poco prima dell’estrazione su una nuova ruota, invia a ciascuno dei socket presenti nella lista il messaggio “INIZIO” e, subito dopo, chiude il socket TCP. Al termine di questa operazione, la lista è azzerata
5. Il client, ottenuto il messaggio, chiude a sua volta il socket TCP e inizia ad acquisire in multicast i numeri delle ruote rimanenti fino a quando riceve, sempre in multicast, il messaggio “FINE”

Caratteristiche del server

Server multithread Java composto da:

- *main thread*: pianifica l’estrazione di una nuova ruota a intervalli regolari avvalendosi della classe *ScheduledExecutorService*; accetta nuove connessioni sulla porta 5070/tcp; aggiunge il socket accettato alla lista dei socket in attesa; attende il completamento delle estrazioni del Lotto da parte del thread di estrazione, quindi procede al salvataggio su file dell’intero tabellone;

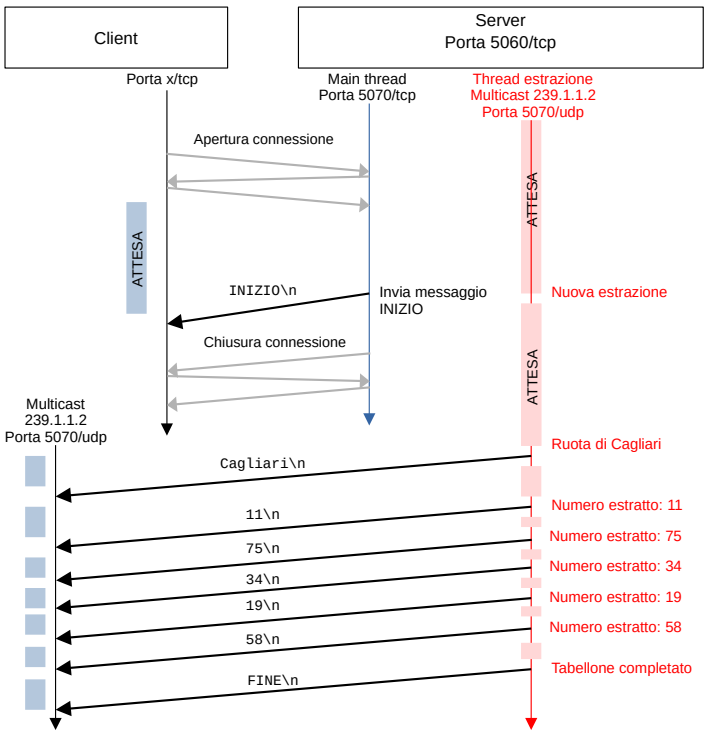
- *thread di estrazione ruota*: esegue la routine di estrazione ripetutamente secondo i vincoli temporali imposti dal main thread (a ogni ripetizione, estrae i numeri di una nuova ruota); invia a ogni socket presente nella lista d’attesa il messaggio “INIZIO”, chiudendo le connessioni TCP; azzera la lista d’attesa; procede all’estrazione sulla nuova ruota inviando i singoli numeri in UDP all’indirizzo di multicast 239.1.1.2, porta 5070/udp; dopo aver inviato l’ultima ruota del tabellone, il thread invia sempre in multicast il messaggio “FINE”.

Caratteristiche del client

Applicazione console Java, svolge le seguenti attività:

- Apre una nuova connessione sulla porta 5070/tcp del server
- Rimane in attesa del messaggio “INIZIO” sul socket TCP aperto
- Ricevuto il messaggio, chiude la connessione TCP (lato client)
- Dopo aver creato un nuovo oggetto MulticastSocket usando la porta 5070/udp, si unisce al gruppo 239.1.1.2
- Riceve in multicast e stampa a video una linea di testo contenente il nome della ruota e, in seguito, cinque linee di testo indicanti i numeri estratti su quella ruota
- Ripete l’attività precedente fino a quando riceve, al posto del nome della ruota, la parola “FINE”.

Lo scambio di messaggi tra un client e il server la sincronizzazione dei thread del server sono descritti dal seguente diagramma temporale.



Protocollo di comunicazione

Si opta per un protocollo *ad hoc* in cui client e server si scambiano messaggi testuali UTF-8 a singola linea. Ogni messaggio termina con un carattere di *newline*.

Soluzione

Sulla piattaforma Netlab è disponibile la soluzione dell’esercizio Gestione del tabellone del lotto (multicast). La soluzione consiste in un progetto multimodulo realizzato con il software IntelliJ Idea.

Crociera panoramica in battello

Realizzare un'applicazione C/S per la vendita di biglietti per una crociera panoramica in battello da N posti.

- *Il client si collega sulla porta 5080/tcp del server per acquistare un biglietto.*
- *Il server rimane in ascolto a ciclo continuo e risponde con il messaggio "ACCETTATO" se ci sono posti disponibili sul battello; in caso contrario invia la frase "NON ACCETTATO" e chiude la connessione.*
- *Il client invia i seguenti dati: cognome, nome, età in anni.*
- *Il server riceve la richiesta, decrementa il numero di posti disponibili ed emette il biglietto contenente i dati: Destinazione (Arona), cognome, nome, età, prezzo. Il prezzo è determinato secondo il seguente criterio: clienti minorenni = 25,00 Euro, maggiorenni 32,50 Euro. I dati del biglietto sono prima stampati a video e successivamente inviati al client.*
- *Il client, ricevuto il biglietto, stampa i dati contenuti e si mette in attesa di ricevere il segnale di inizio della crociera.*
- *Il server, dopo aver venduto tutti i biglietti disponibili (N in tutto), invia a tutti i client il messaggio "PARTENZA" e chiude le connessioni*

Protocollo di comunicazione

In questo esercizio i messaggi scambiati sono composti da oggetti Java serializzabili. In particolare, il server definisce e condivide con il client le classi **Richiesta** e **Biglietto**. Poiché la classe Java *String* è serializzabile, è possibile usare questo metodo anche per inviare semplici stringhe (non serve in questo caso terminare un oggetto “stringa” con *newline).*

Soluzione

Sulla piattaforma Netlab è disponibile la soluzione dell’esercizio Crociera in battello. La soluzione consiste in un progetto multimodulo realizzato con il software IntelliJ Idea.

Appendice

A1 – Generatore di numeri casuali distinti

In alcune applicazioni è richiesta la creazione di sequenze di numeri casuali distinti, cioè sequenze prive di valori duplicati. Per generare numeri casuali distinti, è possibile utilizzare la libreria *Unique Random*, scaricabile in formato JAR all’indirizzo:

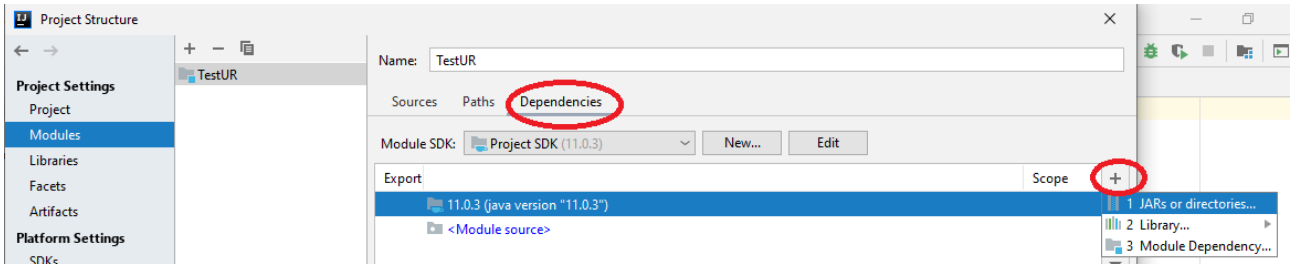
https://www.netlab.fauser.edu/risorse/classi5/tp/UniqueRandom.jar

La libreria include la classe *UniqueRandom* specializzata nella generazione di numeri pseudocasuali interi a 32 oppure 64 bit, diversi uno dall’altro, presi all’interno di un intervallo specificato.

La principale caratteristica di *UniqueRandom* è quella di utilizzare un algoritmo efficiente in memoria in quanto, contrariamente a quanto avviene nella maggior parte dei programmi che creano sequenze pseudocasuali a valori distinti, non richiede l’allocazione di un blocco di memoria di dimensioni pari all’ampiezza dell’intervallo indicato. La complessità spaziale dell’algoritmo implementato in *UniqueRandom* è O(1) , mentre quella degli algoritmi ad allocazione di memoria è solitamente O(*n*), essendo *n* l’ampiezza dell’intervallo: ciò rende *UniqueRandom* utile nella fase di generazione di sequenze molto lunghe.

Per utilizzare la libreria *UniqueRandom* all’interno di un progetto *IntelliJ Idea*, eseguire le seguenti operazioni:

1. aprire il progetto, quindi selezionare dal menu **File** il comando **Project Structure...**
2. selezionare la voce **Modules**, quindi fare click su **Dependencies** e successivamente sul pulsante **+** posto parte a destra della finestra; selezionare infine **JARs or directories...**
3. selezionare il file *UniqueRandom.jar* precedentemente scaricato e confermare la scelta;
4. Premere il pulsante OK per chiudere la finestra *Project Structure*.



A questo punto è possibile utilizzare la classe *UniqueRandom* nel codice del progetto. Per esempio, è possibile simulare l’estrazione di tutti i 90 numeri della tombola scrivendo il codice seguente:

```
import edu.fauser.netlab.UniqueRandom;

public class Main {

    public static void main(String[] args) {
        final int minimo = 1;
```

```
        final int massimo = 90;
        final long seed = 2019;

          UniqueRandom r = new UniqueRandom(minimo, massimo + 1, seed);
        int contaValori = massimo - minimo + 1;
        for (int i = 0; i < contaValori; i++) {
            int n = r.nextInt();
            System.out.format("%4d", n);
        }
        System.out.println();
    }
}
```

Il costruttore della classe *UniqueRandom* accetta tre parametri: il primo specifica il limite inferiore dell’intervallo (incluso), il secondo il limite superiore (escluso) e il terzo rappresenta il seme (*seed*) utilizzato nella generazione dei valori pseudocasuali. Se il terzo parametro è omesso, il seme è ricavato automaticamente dall’orologio di sistema.

I principali metodi della classe *UniqueRandom* sono:

- *nextInt():* genera un nuovo elemento della sequenza pseudocasuale diverso da quelli creati precedentemente; il risultato restituito da questo metodo è un intero con segno a 32 bit. Nel caso in cui l’intervallo specificato contenga numeri non rappresentabili su 32 bit, l’invocazione del metodo *nextInt()* genera un’eccezione;
- *nextLong()*: opera in maniera analoga a *nextInt()* ma restituisce un intero con segno a 64 bit e non genera alcuna eccezione.

A2 – Deployment di un’applicazione Java su server Linux

A2.1 – Installazione del software Java Runtime Environment (JRE)

Per installare il software Openjdk JRE 11 su un server Ubuntu 18.04 eseguire i seguenti comandi:

```
sudo apt update
sudo apt install openjdk-11-jre
```

Al termine dell’installazione verificare la versione di Java con il comando:

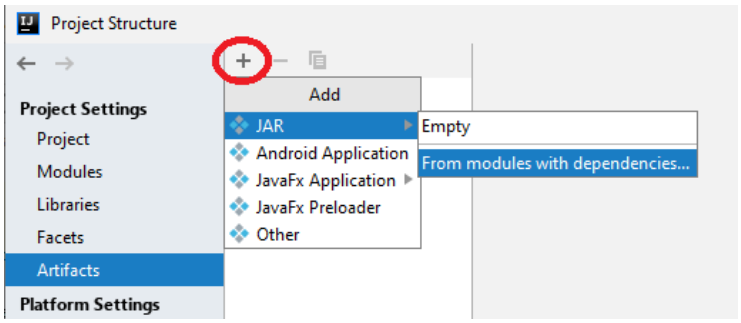
```
java -version
```

A2.2 – Memorizzazione dell’applicazione in formato JAR

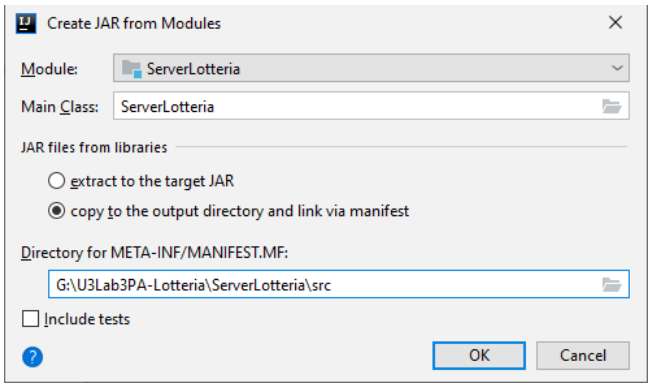
Per consentire la distribuzione di applicazioni Java complesse è preferibile raccogliere tutte le classi sviluppate, le librerie richieste e le risorse relative in un unico archivio costituito da un file di estensione .jar (Java Archive).

Per generare un file .jar all’interno di un progetto sviluppato con IntelliJ Idea:

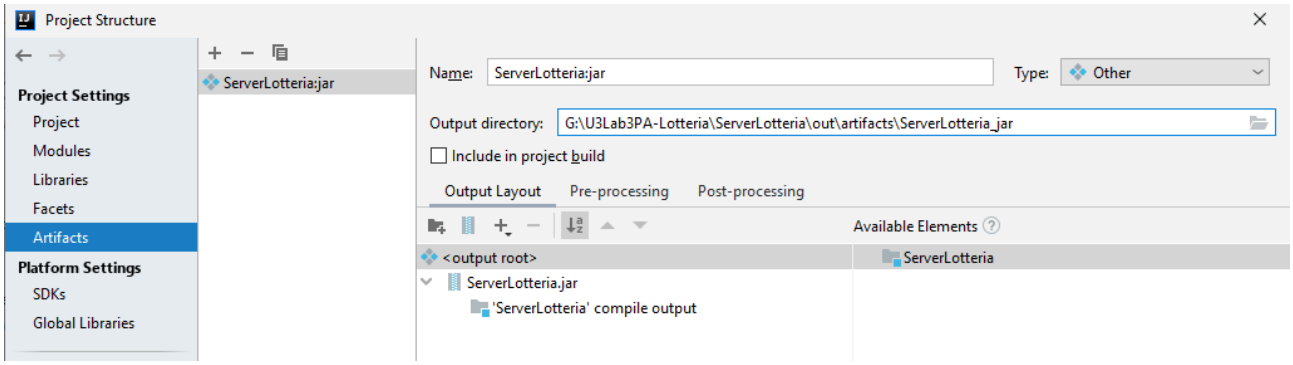
1. Eseguire il comando **File → Project Structure…**, quindi selezionare la voce **Artifacts**;
2. Aggiungere un nuovo file .jar premendo il pulsante **+** , quindi selezionando **JAR → From modules with dependencies**;



3. Nella finestra *Create JAR from Modules* indicare la classe contenente il metodo *main*, quindi selezionare l’opzione “*Copy to the output directory and link via manifest...*” e specificare la directory che conterrà il file di manifesto (in generale coincidente con la cartella che contiene i sorgenti Java dell’applicazione);



4. Confermare le impostazioni del file .jar premendo il pulsante *Ok*.



Per generare il file .jar contenente le classi dell’applicazione, eseguire dal menù principale di IntelliJ Idea il comando:
Build → Build Artifacts… (action: Build).

Il nuovo archivio in formato JAR è memorizzato nella sottocartella *out\artifacts\nomeapp_jar*. Questa operazione deve essere ripetuta ogni volta che si modificano i file sorgenti.

A2.3 – Trasferimento dell’applicazione sul server

Copiare sul server Linux il file .jar generato precedentemente utilizzando il software WinSCP.
Successivamente, accedere al server usando un account con adeguati privilegi ed eseguire i seguenti comandi (nell’esempio il nome dell’account è **utente**, mentre il nome del file .jar è **ServerLotteria.jar**):

1. Spostamento del file .jar nella cartella /opt:

   ```
   sudo mkdir /opt/server-lotteria
   sudo mv /home/utente/ServerLotteria.jar /opt/server-lotteria
   ```

2. Preparazione del file .service (necessario per eseguire automaticamente l’applicazione ogni volta che si avvia il server)

   ```
   sudo nano /etc/systemd/system/server-lotteria.service
   ```

   Inserire nel nuovo file le seguenti righe di testo:

```
[Unit]
Description=Applicazione "Lotteria"

[Service]
User=utente
WorkingDirectory=/opt/server-lotteria
ExecStart=/usr/bin/java -Xms128m -Xmx256m -jar ServerLotteria.jar
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Salvare le modifiche, quindi uscire dall’editor.

3. Avviare il servizio “server-lotteria”:

```
sudo systemctl daemon-reload
sudo systemctl start server-lotteria.service
sudo systemctl status server-lotteria.service
```

4. Abilitare l’esecuzione automatica del servizio all’avvio del server:

```
sudo systemctl enable server-lotteria.service
```

5. Eseguire alcuni test per verificare il corretto funzionamento del nuovo servizio erogato dal server (con Putty oppure con un adeguato client)

Applicazioni proposte

I. [Progetto **ClientTraduttore**] – Un servizio di traduzione di semplici testi[1], erogato dall'host *services.netlab.fauser.edu*, è attivo sulla porta TCP 23131.

A livello applicativo, il client invia una richiesta al server in formato testuale (codifica UTF-8) contenente la frase da tradurre e un insieme di informazioni di supporto secondo il seguente formato:

```
lingua1/frase1/lingua2<EOL>
```

in cui:

(a) *lingua1*: è il codice ISO 639-1 della lingua in cui è stata scritta la frase da tradurre;

(b) *frase1*: è la frase oggetto della traduzione (se la frase contiene il carattere '/', tale carattere deve essere codificato in "//");

(c) *lingua2*: è il codice ISO 639-1 della lingua in cui sarà riportata la traduzione.

Per conoscere i codici delle lingue supportate dal servizio di traduzione, consultare la pagina https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

I dati della richiesta sono riportati su un'unica linea e separati dal carattere '/'. La linea deve terminare con il simbolo EOL (End Of Line, corrispondente al carattere di *newline* '\n').

Il server, ricevuta la richiesta, esegue la traduzione e restituisce il risultato al client secondo il seguente formato:

```
lingua1/frase1/lingua2/frase2<EOL>
```

dove "*frase2*" rappresenta il risultato della traduzione nella lingua richiesta. Nel caso in cui non sia possibile eseguire la traduzione, il server invia un messaggio d'errore nel formato:

```
ERR/messaggio<EOL>
```

Scrivere un'applicazione che consenta a un utente di tradurre un testo dall'italiano all'inglese. L'applicazione richiede l'inserimento da tastiera di una frase in italiano, invia un'opportuna richiesta al server e, dopo aver ricevuto la risposta, visualizza sullo schermo la frase tradotta in inglese oppure, in caso di problemi di traduzione, il corrispondente messaggio d'errore. L'operazione è ripetuta fino a quando l'utente immette una stringa vuota.

È richiesta un'adeguata elaborazione dei dati affinché siano nascosti all'utente finale i dettagli della struttura dei messaggi scambiati (codici ISO delle lingue, caratteri separatori, ecc.).

II. [Progetto **ClientMultilingua**] – A partire dal progetto precedente, realizzare un'applicazione client multilingua che permetta all'utente di scegliere le lingue in cui scrivere e tradurre le frasi. Le lingue sono scelte mediante un apposito menù contenente almeno cinque lingue (tra cui l'italiano) supportate dal servizio.

1 L’applicazione che eroga il servizio è stata sviluppata in C# nell’ambito delle attività di laboratorio di *Sistemi e reti*. Gli studenti interessati a conoscere il funzionamento del servizio possono richiedere al docente il codice sorgente dell’applicazione.

III. [Progetto **AnalisiChimica**] – Realizzare un'applicazione C/S in grado di eseguire automaticamente l'analisi chimica di un certo composto. A partire dalla formula chimica del composto (per esempio H2O), l'applicazione deve determinare la percentuale degli elementi presenti (88,81% di ossigeno e 11,19% di idrogeno) tenendo conto del numero di atomi e dei rispettivi pesi atomici.

Il server eroga il servizio di analisi[2] sulla porta 5120/tcp e svolge le seguenti attività.

(a) Riceve la formula chimica del composto da cui ricava i simboli degli elementi e il numero di atomi di ogni elemento (per esempio, due atomi di idrogeno e uno di ossigeno). Se la formula ricevuta non è corretta, il server restituisce un apposito messaggio d'errore.

(b) Calcola il peso molecolare del composto a partire dai pesi atomici degli elementi rilevati, quindi determina la percentuale di ogni singolo elemento[3]. Il server dispone del file di testo *elementichimici.txt* contenente il simbolo chimico, il nome e il peso atomico di tutti gli elementi.

(c) Restituisce un messaggio di risposta contenente:

- il nome del server che ha eseguito l'operazione e la data dell'analisi
- l'elenco delle percentuali degli elementi, in ordine descrescente

Si riporta di seguito una possibile risposta prodotta dal server:

```
Analisi chimica eseguita dal computer SERVER01 in data 2021/11/30 16:21:37 CET
Composto analizzato: H2O
 88,81%  Ossigeno (O)
 11,19%  Idrogeno (H)
```

Il client legge da tastiera un stringa contenente la formula chimica da analizzare, apre una connessione al server sulla porta specificata, invia la formula, visualizza l'esito dell'analisi e chiude la connessione. Il client ripete le precedenti attività fino a quando l'utente inserisce una stringa vuota.

2 Il servizio di analisi chimica è anche disponibile online all’indirizzo *services.netlab.fauser.edu (*porta 5120/tcp). Il servizio è erogato da un cluster di server.

3 Il peso atomico dell’idrogeno è *pa(H)* = 1.008, quello dell’ossigeno *pa(O)* = 15.999. La molecola d’acqua è composta da due atomi di idrogeno e uno di ossigeno, quindi il suo peso molecolare è:
pm = 2**pa(H)* + 1**pa(O)* = 2*1.008 + 1*15.999 = 18.015.
La percentuale di idrogeno è pertanto pari a 2**pa(H)* / *pm* * 100% = 2*1.008 / 18.015 * 100% = 11.19%, quella di ossigeno 1**pa(O)* / *pm* * 100% = 1*15.999 / 18.015 * 100% = 88.81%