ESERCIZIO 19 P. 88-89 CON UDP

19 DESIGN CODING Nello sport dell'orienteering i concorrenti dimostrano il passaggio dai punti di tappa acquisendo con l'APP del proprio smartphone il QR-code esposto: l'APP invia l'identificativo della tappa costituito da una stringa alfanumerica e quello numerico del concorrente registrato nell'APP a un server UDP
Dopo aver definito e documentato un protocollo testuale applicativo che permetta l'invio al server dell'identificativo numerico del concorrente e di quello alfanumerico del QR-code di una tappa, progettare mediante un diagramma UML e implementare in linguaggio Java una classe che consenta lato server di registrare gli identificativi concorrente/tappa ricevuti associandoli a un *timestamp* rappresentante il numero di secondi trascorsi dalle 00:00:00 del 1/1/1970.
Realizzare infine un server UDP in linguaggio Java che utilizzi la classe implementata per la registrazione dei dati ricevuti.

Realizzare una classe client UDP che consenta di testare il server

SOLUZIONE

Definizione del protocollo:

Il codice identificativo alfanumerico della tappa è costituito da un codice di tre lettere che identifica il luogo di svolgimento della gara seguito dalla lettera "T" che indica la tappa, seguito dal numero progressivo della tappa. Ad esempio

DART1 → Darfo Tappa 1

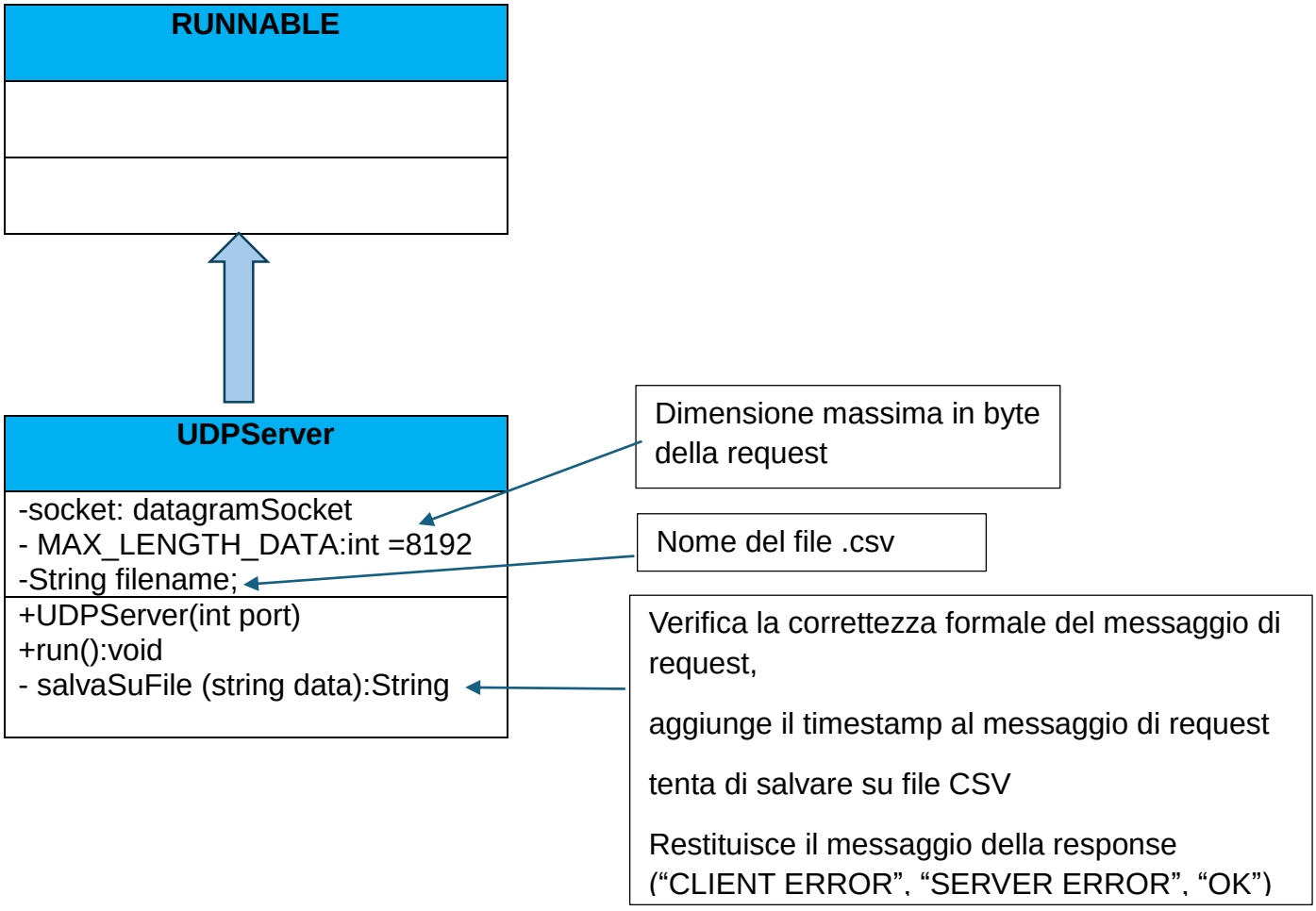
DART2 → Darfo Tappa 2

…..

Il codice identificativo del concorrente è costituito da un numero intero che indica il numero di iscrizione del concorrente.

Request del client	Azione del server	Risposta del server
codice identificativo della tappa;codice identificativo del concorrente **esempio:** **DART1;1**	Il server memorizza IN APPEND su un file CSV i dati contenuti nella request aggiungendo il timestamp Esempio: DART1;1;1735732822	**OK se la memorizzazione va a buon fine**
		CLIENT ERROR se il formato della request non è corretto
		SERVER ERROR se la memorizzazione non va a buon fine

DIAGRAMMA UML



Coidice

```
public class UDPServer implements Runnable
{
    private DatagramSocket socket;
    private final int MAX_LENGTH_DATA=8192;
    private     String filename="orienteering.csv";

    public UDPServer(int port) throws SocketException
    {
        //istanzio il socket per leggere/scrivere dati sulla prota "port"
        socket=new DatagramSocket(port);
        //assegno un timeout di 1 s, dopo un secondo il socket si chiude e poi si rimetterà in
ascolto
        socket.setSoTimeout(1000);
    }

    //Verifico se il messaggio è formalmente corretto
    private boolean checkMessageRequest(String requestMessage)
    {
        String[] requestElements;
        String codiceTappa;
        String stringNumeroTappa;
        int idConcorrente;
        int numeroTappa;

        requestElements=requestMessage.split(";");
        codiceTappa=requestElements[0];

        if (codiceTappa.charAt(0)<'A' || codiceTappa.charAt(0)>'Z')
            return false;
        if (codiceTappa.charAt(1)<'A' || codiceTappa.charAt(1)>'Z')
            return false;
        if (codiceTappa.charAt(2)<'A' || codiceTappa.charAt(2)>'Z')
            return false;
        if (codiceTappa.charAt(3)!='T')
            return false;
        if (codiceTappa.charAt(3)!='T')
            return false;
        //Estraggo il numero tappa dal codiceTappa per verificare se è un numero
        stringNumeroTappa=codiceTappa.substring(4, codiceTappa.length());
        try
        {
            //verifico che il numero di tappa e l'idConcorrente siano numerici
            numeroTappa=Integer.parseInt(stringNumeroTappa);
            idConcorrente=Integer.parseInt(requestElements[1]);
        }
        catch( NumberFormatException ex)
        {
            return false;
        }
        return true;
```

```
    }

    String salvaSuFile(byte[] requestData,int dataLength)
    {

        String requestMessage;
        try
        {
            requestMessage = new String(requestData,"UTF-8");
        }
        catch (UnsupportedEncodingException ex)
        {
           return "CLIENT ERROR"; //request mal formata poichè contiene cratteri non UTF-8
        }
        //Tagliamo la request ai soli caratteri effettivamente presenti nella request
        requestMessage=requestMessage.substring(0,dataLength);
        //controllo la correttezza della request string
        if (!checkMessageRequest(requestMessage))
            return "CLIENT ERROR"; //request mal formata

        //aggiungo il timestam al requestMessage e salvo su file

requestMessage=requestMessage+";"+Long.toString(Instant.now().getEpochSecond());
        try
        {
            TextFile file=new TextFile(filename, 'W', true);
            file.toFile(requestMessage);
            file.close();
        }
        catch (IOException ex)
        {
            return "SERVER ERROR"; //Impossibile salvare sul file
        }
        catch (FileException ex)
        {
            return "SERVER ERROR"; //Impossibile salvare sul file, file aperto in lettura
        }
        return "OK";
    }

    public void run()
    {
        byte[] data=new byte[MAX_LENGTH_DATA];
        //istanzio un datagramma vuoto che ospiterà il datagarmma ricevuto
        DatagramPacket request=new DatagramPacket(data, data.length);
        //reference al datagramma di risposta
        DatagramPacket response;
        String responseMessage;
        while(!Thread.interrupted())
        {
            try
```

```
            {
                //si mette in ascolto sulla prota "port" in attesa di un datagramma
                //dopo un secondo interromperà l'ascolto e poi si rimetterà in ascolto (Timeout)
                socket.receive(request);
                //verifico la correttezza della request ed eventualmente salvo su file i dati,
                //il valore restituito è il messaggio da inserire nella response
                responseMessage=salvaSuFile(request.getData(), request.getLength());

                response=new DatagramPacket(responseMessage.getBytes("UTF-8"),
responseMessage.length(),request.getAddress(),request.getPort());
                //invio la response
                socket.send(response);
            }

            catch (SocketTimeoutException ex)
            {
                System.out.println("Timeout");
            }
            catch (IOException ex)
            {
                System.out.println("Errore di comuniczione");
            }
        }
        socket.close();
        System.out.println("Server interrotto");
    }

    public static void main(String[] args)
    {
        Scanner tastiera=new Scanner(System.in);
        try
        {

            UDPServer echoServer=new UDPServer(7);
            Thread echoServerThread=new Thread(echoServer);
            echoServerThread.start();
            tastiera.nextLine();
            echoServerThread.interrupt();
        }
        catch (SocketException ex)
        {
            System.out.println("Impossibile aprire il socket del server");
        }
    }

}
```

```
public class UDPClient
{
   private DatagramSocket socket;

   public UDPClient() throws SocketException
   {
      //Istanzio il socket del client e
      //assegno un Timeout di 1 s
      socket=new DatagramSocket();
      socket.setSoTimeout(1000);
   }

   public String sendAndReceive(String message, String host, int port) throws
SocketTimeoutException,UnknownHostException, UnsupportedEncodingException,
IOException
   {
      DatagramPacket request;
      DatagramPacket response;
      byte[] responseData=new byte[8192];
      String responseMessage;

      InetAddress ipAddress=InetAddress.getByName(host);
      request=new DatagramPacket(message.getBytes("UTF-8"),
message.length(),ipAddress,port);
      response=new DatagramPacket(responseData, responseData.length);
      socket.send(request);
      socket.receive(response);

      if (response.getAddress().equals(ipAddress) && response.getPort()==port)
      {
         responseMessage=new String(response.getData(),"UTF-8");
         //"taglio la risposta ai soli caratteri utili
         responseMessage=responseMessage.substring(0,response.getLength());
      }
      else
      {
         responseMessage="Error!";
      }


      return responseMessage;
   }

   public void close()
   {
      socket.close();
   }

   public static void main(String[] args)
   {
      String ipServer="127.0.0.1";
```

```
        int port=7;
        String message="DART1;34";

        /*
           IL MESSAGGIO PUO' ESSERE FATTO INSERIRE COME DATO DI INPUT
DALL'UTENTE
        */


        UDPClient echoClient = null;

        String serverAnswer;

        try
        {
           echoClient=new UDPClient();
           serverAnswer=echoClient.sendAndReceive(message, ipServer, port);
           System.out.println("Risposta del server --> "+serverAnswer);
        }
        catch (SocketException ex)
        {
           System.out.println("Impossibile aprire il socket del client");
        }
        catch (UnknownHostException ex)
        {
            System.out.println("Host non raggiungibile");
        }
        catch (UnsupportedEncodingException ex)
        {
            System.out.println("Messaggio non corretto");
        }
        catch (SocketTimeoutException ex)
        {
          System.out.println("Il server non risponde");
        }
        catch (IOException ex)
        {
          System.out.println("Errore di comunicazione");
        }

        echoClient.close();
    }
}
```