



Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita Client-server con socket – uso della serializzazione

Luigi Lavazza

Dipartimento di Scienze Teoriche e Applicate

luigi.lavazza@uninsubria.it



Premessa

- Gli esercizi sono gli stessi dell'esercitazione precedente.
- In questo caso però usiamo la serializzazione per inviare oggetti su socket.



Esercizio 1 – Segment server

- Nel codice sorgente dato, una classe **SegmentClient** utilizza una classe **Segment** locale
- La classe **Segment** a sua volta usa la classe **Point**



class Point

```
public class Point {  
    double x, y;  
    final double THRESHOLD = 0.0000001;  
    public Point(double tx, double ty){  
        x=tx; y=ty;  
    }  
    public double getX(){  
        return x;  
    }  
    public double getY(){  
        return y;  
    }  
    public boolean isEqual(Point p){  
        return (Math.abs(x - p.getX()) < THRESHOLD) &&  
            (Math.abs(y - p.getY()) < THRESHOLD) ;  
    }  
}
```



class Segment

```
public class Segment {  
    private Point p1=null;  
    private Point p2=null;  
    public Segment(){ }  
    public boolean set(Point p1, Point p2) {  
        if(!p1.isEqual(p2)) {  
            this.p1=p1; this.p2=p2;  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```



class Segment

```
private Point midPoint() {
    double mx, my;
    mx = (p1.getX() + p2.getX()) / 2;
    my = (p1.getY() + p2.getY()) / 2;
    return (new Point(mx, my));
}

public Point simmetric(Point p) {
    Point m = this.midPoint();
    double simmX = 2 * m.getX() - p.getX();
    double simmY = 2 * m.getY() - p.getY();
    return new Point(simmX, simmY);
}
}
```



main

```
public class SegmentClient {  
    public static void main(String[] args) {  
        Point p1=new Point(0.0, 0.0);  
        Point p2=new Point(4.0, 4.0);  
        Point px=new Point(0.0, 4.0);  
        Segment sgm=new Segment();  
        sgm.set(p1, p2);  
        Point simm = sgm.simmetric(px);  
        System.out.println("Il punto simmetrico e` (" +  
            +simm.getX()+" , "+simm.getY()+" )");  
    }  
}
```

Esercizio 1

- Si vuole realizzare un server che sia in grado di accettare connessioni **via socket** e che implementi le seguenti richieste da parte dei client:
 - 1) creazione di segmento.
 - In questo caso un client scrive sul socket il comando "NewSegment" seguito da due oggetti di classe Point che rappresentano gli estremi del segmento.
 - Il server risponde "OK" o "KO" a seconda che l'operazione sia riuscita o meno.
 - 2) trovare il punto simmetrico, rispetto al segmento, di un punto dato.
 - Il client scrive sul socket il comando "Simmetrico" seguito dall'oggetto di classe Point di cui si vuole trovare il simmetrico.
 - Il server risponde scrivendo sul socket l'oggetto di classe Point richiesto.
 - Si implementi il server descritto.
 - Si vuole che molti client possano accedere contemporaneamente al server, ma ciascuno per creare e usare un proprio segmento (diverso da quello degli altri client).
-



Esercizio 1

- Nella esercitazione precedente, non avendo ancora visto la serializzazione, client e server si scambiavano solo stringhe di caratteri.
- Adesso invece vogliamo implementare un soluzione che non soffre di tale limitazione: client e server si scambiano oggetti serializzati.
 - ▶ NB: questo implica che il client deve conoscere e usare la classe Point, e che client e server devono usare la medesima versione della classe



Esercizio 2 – asta on-line semplificata

- Un server gestisce un'asta.
- Il server accetta connessioni dai client, che possono fare offerte.
- L'asta si conclude quando il server non riceve connessioni per un certo tempo
 - ▶ Vedere setTimeout

Esercizio – asta on-line semplificata

- I client si connettono e possono emettere i seguenti comandi:
 - ▶ «quit»: fine della connessione
 - ▶ «read»: richiesta di conoscere il valore corrente offerto per il bene all'asta
 - A seguito di questo comando, il server manda al client il valore dell'offerta che al momento si aggiudicherebbe il bene messo all'asta
 - ▶ «offer»: offerta. Questo comando è seguito da un valore intero (la somma offerta per il bene all'asta) e da una stringa (il nome dell'offerente). A seguito di questo comando, il server manda al client il risultato dell'offerta:
 - «OK» indica che l'offerta è stata accettata,
 - «KO» indica che è stata rifiutata (perché inferiore al minimo rilancio richiesto, oppure perché nel frattempo qualcuno ha offerto di più).



Esercizio – asta on-line semplificata

- Perché l'asta è «semplificata»?
- Perché in un'asta vera il server dovrebbe notificare a tutti i client connessi gli avvenimenti salienti
 - ▶ accettazione di un'offerta da parte di un altro client,
 - ▶ chiusura dell'asta.
- Per fare questo si usano tecniche che vedremo più avanti.



`setSoTimeout` (class `Socket`)

`public void setSoTimeout(int timeout)`

`throws SocketException`

- Enable/disable `SO_TIMEOUT` with the specified timeout, in milliseconds. With this option set to a non-zero timeout, a `read()` call on the `InputStream` associated with this `Socket` will block for only this amount of time. If the timeout expires, a `java.net.SocketTimeoutException` is raised, though the `Socket` is still valid. The option must be enabled prior to entering the blocking operation to have effect. The timeout must be > 0 . A timeout of zero is interpreted as an infinite timeout.
- Parameters:
 - ▶ `timeout` - the specified timeout, in milliseconds.
- Throws:
 - ▶ `SocketException` - if there is an error in the underlying protocol, such as a TCP error.



Esercizio 3 – Produttori e consumatori distribuiti

- Si realizzi un sistema distribuito in cui
 - ▶ I clienti sono produttori e consumatori
 - ▶ Il server gestisce il buffer condiviso
- Ciò che viene prodotto e consumato sono istanze della classe **Roba**, che contiene un identificatore, un tipo, una descrizione, ecc.