Programmazione Avanzata 01/07/2022

Note sullo svolgimento della prova

- Non è consentito l'uso di vecchi testi d'esame, libri, o appunti.
- Utilizzare NetBeans o Notepad++ come editor, e finestre DOS per compilare ed eseguire i programmi.
- La cartella C:\prg contiene il JDK da utilizzare e la documentazione delle Java API. Eventuali file ausiliari preparati dal docente devono essere scaricati da elearn.ing.unipi.it
- I file sorgenti relativi all'esercizio 1 devono essere posti nella cartella C:\prg\myapps\esercizio1 mentre quelli relativi all'esercizio 2 devono essere posti nella cartella C:\prg\myapps\esercizio2.
- Al termine della prova, dopo il caricamento degli elaborati su elearn.ing.unipi.it, il docente mostrerà una possibile soluzione. Dopo aver visto la soluzione, gli studenti avranno la possibilità di riprendere il proprio elaborato.

Esercizio 1

Una sala di attesa può contenere un numero massimo di pazienti pari a N. Quando si raggiunge tale numero massimo, non viene permesso nessun nuovo ingresso finché il numero di pazienti all'interno della sala non diventa pari a N − K, dove 0 ≤ K < N. I pazienti sono caratterizzati da un livello di priorità, espresso come un numero intero tra 1 e 5. Il valore 1 esprime la priorità massima, 5 quella minima. In caso di competizione, l'accesso alla sala avviene secondo il livello di priorità.

Realizzare una classe *Sala* con almeno i seguenti metodi:

- *Sala(int N, int K)*: costruisce un oggetto *Sala* dalle caratteristiche specificate; lancia *ParametriErratiException* se i parametri non sono validi;
- *void entrata(int prio)*: un paziente chiede di entrare nella *Sala* specificando il proprio livello di priorità; l'operazione può essere bloccante;
- *void uscita()*: un paziente notifica la propria uscita dalla sala.

Realizzare una classe *Paziente* che si comporta come segue: entra nella sala, rimana nella sala un tempo casuale compreso tra 1 e 2 secondi, esce dalla sala.

Realizzare una classe di prova che crea 10 pazienti con livelli di priorità a caso. Esempio di output:

```
Paziente Thread-0 con codice 2 entrato
Paziente Thread-9 con codice 1 entrato
Paziente Thread-8 con codice 1 entrato
Paziente Thread-3 con codice 1 entrato
Paziente Thread-6 con codice 0 entrato
Sala chiusa
Paziente Thread-7 con codice 3 in attesa
Paziente Thread-5 con codice 1 in attesa
Paziente Thread-4 con codice 2 in attesa
Paziente Thread-2 con codice 1 in attesa
Paziente Thread-1 con codice 4 in attesa
Paziente Thread-8 uscito
Paziente Thread-3 uscito
Sala riaperta
Paziente Thread-7 con codice 3 in attesa
Paziente Thread-1 con codice 4 in attesa
Paziente Thread-2 con codice 1 entrato
Paziente Thread-4 con codice 2 in attesa
Paziente Thread-5 con codice 1 entrato
Sala chiusa
Paziente Thread-0 uscito
Paziente Thread-5 uscito
Sala riaperta
Paziente Thread-7 con codice 3 entrato
Paziente Thread-4 con codice 2 entrato
Sala chiusa
Paziente Thread-1 con codice 4 in attesa
Paziente Thread-6 uscito
Paziente Thread-2 uscito
Sala riaperta
Paziente Thread-1 con codice 4 entrato
Paziente Thread-7 uscito
Paziente Thread-9 uscito
Paziente Thread-4 uscito
Paziente Thread-1 uscito
```

Esercizio 2

Un sistema di *crowd answering* (“risposta della folla”) consente di fornire una risposta adeguata ed economica a una domanda, sulla base della somma di tante esperienze particolari, più di quanto si possa ottenere da un singolo esperto sulla base di una conoscenza generale. Si consideri un numero di *Opinionisti* non noto a priori, in comunicazione secondo una configurazione ad anello. Un *Opinionista* (interrogante) formula una *Domanda*, esprime la propria opinione, la visualizza e la invia al successivo *Opinionista*. Si supponga per semplicità che la opinione sia un numero reale. Una domanda è composta da: *testo*, *somma delle opinioni*, *numero di opinioni, numero sufficiente di opinioni*, e *porta di ascolto dell’opinionista interrogante*. Ogni *Opinionista* riceve la domanda, dà una opinione, la visualizza e la invia all’*opinionista* successivo, e cosi via. Durante il primo giro, non appena si giunge al numero sufficiente di opinioni (anche se questo dovesse avvenire con una sola opionine) la domanda viene inviata all’opinionista interrogante, il quale visualizza la domanda e la risposta (calcolata come valor medio delle opinioni), la invia all’opinionista successivo, e cosi via per il secondo giro. Il secondo giro viene effettuato solo se c’è una risposta, e serve a ciascun opinionista a ricevere, visualizzare la domanda e la risposta, inviare la domanda, fino a fermarsi all’opinionista che per primo ha precedentemente verificato il numero sufficiente di opinioni, il quale rimanderà la domanda all’opinionista interrogante per concludere il giro.

Si realizzi un’applicazione Java distribuita composta dalle classi *Domanda* e *Opinionista*. La classe *Domanda* mantiene ed elabora i dati suddetti. La classe *Opinionista* svolge le operazioni di trasmissione di istanze di *Domanda* tra un opinionista e il successivo, in accordo al protocollo di cui sopra. Si assuma che le visualizzazioni, gli invii e le ricezioni avvengano in formato XML, e che le altre operazioni avvengano su istanze di Domanda. **Attenersi esattamente a quanto espresso dalle seguenti figure**. Fig.1 mostra una configurazione di test con soli tre opinionisti. Fig.2 mostra i file dell’applicazione. Fig.3 presenta le classi da realizzare (in grigio), con i relativi campi e i metodi da creare, i package e le classi da importare e usare. Gestire la chiusura di flussi e connessioni aperti. Catturare solo le eccezioni obbligatorie, e gestirle semplicemente stampando le relative informazioni. Fig.4 mostra il file di testing make.bat, con alcuni casi di test. Si noti che l’opinionista interrogante si distingue per avere in ingresso più di tre parametri. La logica di *Domanda* e Opinionista dovrà essere valida a prescindere dai numeri di porta e di opinionisti.

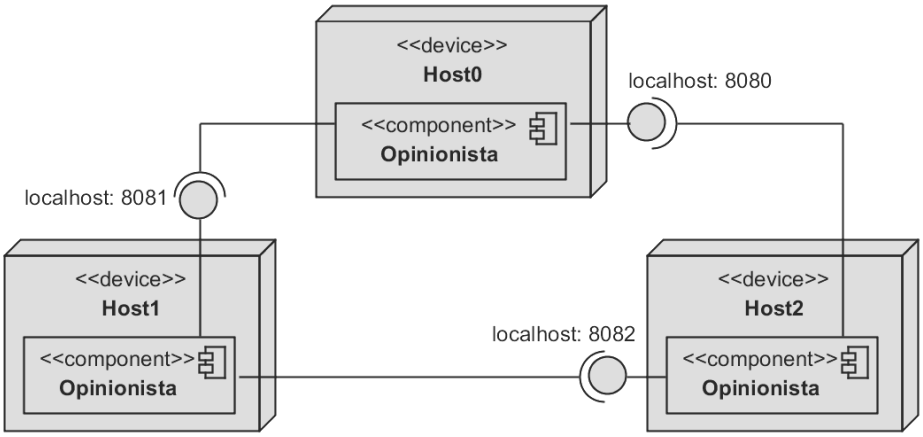
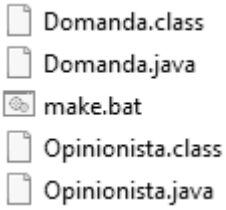


Fig.1 - Tre istanze di *Opinionista* e relative porte di ascolto



(viene fornito solo make.bat)

Fig.2 - c:\prg\esercizio2

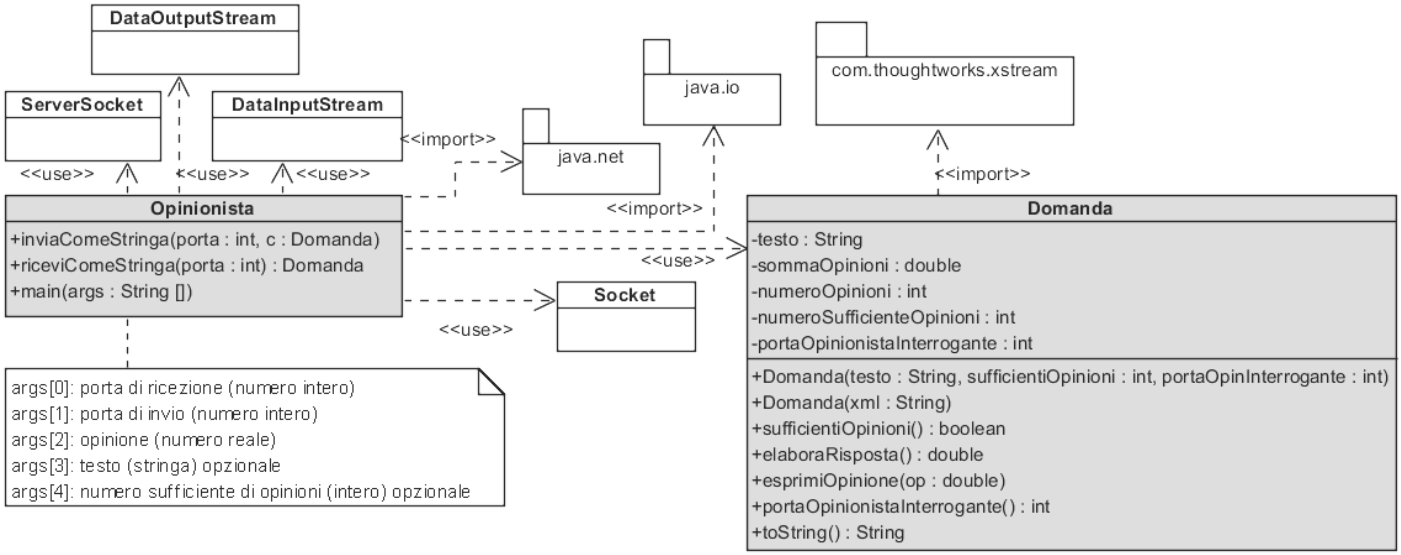


Fig.3 - Classi di cui deve essere composta l’applicazione (in grigio) e classi/package da usare (in bianco)

```
@echo off
c:\prg\jdk8\bin\javac -classpath c:\prg\libs\xstream-1.4.7.jar *.java
pause

set L=c:\prg\libs\
set LIBS=%L%xstream-1.4.7.jar;%L%xmlpull-1.1.3.1.jar;%L%xpp3_min-1.1.4c.jar;

start cmd /k "color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8082 8080 50"
start cmd /k "color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8081 8082 60"
start cmd /k "color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8080 8081 55 "Altezza
              della Torre di Pisa" 2"
pause
taskkill /f /im "java.exe"
rem risultato 57.5  calcolato senza il contributo di 8082
pause

start cmd /k "color 3F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8082 8080 50"
start cmd /k "color 3F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8081 8082 60"
start cmd /k "color 3F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8080 8081 55 "Altezza
              della Torre di Pisa" 3"
pause
taskkill /f /im "java.exe"
rem come risultato 55 calcolato con il contributo di tutti
pause

start cmd /k "color 4F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8082 8080 50"
start cmd /k "color 4F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8081 8082 60"
start cmd /k "color 4F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8080 8081 55 "Altezza
              della Torre di Pisa" 4"
pause
taskkill /f /im "java.exe"
rem nessun risultato (numero minimo non raggiunto)
pause

start cmd /k "color 5F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8082 8080 50"
start cmd /k "color 5F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8081 8082 60"
start cmd /k "color 5F && c:\prg\jdk8\bin\java -classpath %LIBS% Opinionista 8080 8081 55 "Altezza
              della Torre di Pisa" 1"
pause
taskkill /f /im "java.exe"
rem risultato 55 calcolato con il solo contributo di 8080
pause
```

Fig.4 - File make.bat, da non modificare.