Programmazione 21/2/2019

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 la sottomissione 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

Realizzare le classi *Roulette*, *Giocatore, Prova* e eventuali altri classi ausiliarie secondo le seguenti specifiche.

Classe Roulette:

- *Roulette()*: crea un oggetto Roulette con tempo minimo pari a 1000 e tempo massimo pari a 4000.
- *Roulette(long tempoMin, long tempoMax)*: crea un oggetto Roulette caratterizzato dai tempi minimi e massimi specificati. I tempi sono espressi in ms.
- *void avvia()*: quando viene chiamato la ruota e la pallina cominciano a girare. La pallina si sposta su un nuovo numero casuale (compreso tra 0 e 36) ogni 0.1 secondi. La pallina e la ruota girano per un tempo casuale uniformemente distribuito tra il tempo minimo e il tempo massimo. I giocatori possono eseguire puntate prima che la pallina e la ruota comincino a girare, o mentre sono in movimento, ma solo fino a un tempo pari a metà del tempo totale per cui pallina e ruota devono girare.
- *boolean puntata(int n) throws PuntateNonPossibiliException*: chiamato dai giocatori per puntare sul numero *n*. Il metodo lancia *PuntateNonPossibiliException* se: i) la partita è già terminata; ii) la pallina e la ruota stanno ancora girando ma è passato più della metà del tempo per cui devono girare.

Classe *Giocatore*: sceglie un numero casuale compreso tra 1 e 36 ed esegue una puntata dopo aver atteso per un tempo casuale.

Classe *Prova*: crea una roulette, 5 giocatori e avvia il gioco.

Esempio di esecuzione:

```
36
8
0
24
Thread-1: punto sul 31
15
6
29
Thread-4: punto sul 29
30
6
Rien ne va plus, les jeux sont faits!
32
27
2
Thread-0: punto sul 21
Thread-0: troppo tardi...
33
26
25
29
Thread-4: ho vinto
Thread-1: non ho vinto
Thread-2: punto sul 4
Thread-2: troppo tardi...
Thread-3: punto sul 36
Thread-3: troppo tardi...
```

Esercizio 2

La classe *PCList (Persistent Circularly Linked List*) gestisce una sequenza di valori reali, suddivisa in sotto-sequenze su un numero non predefinito di host comunicanti secondo uno schema ad anello (es. Fig.1). Per gestire in modo coordinato le sotto-sequenze, si adopera una istanza di *PCList* per ognuna di esse. La classe *PCList* non mantiene lo stato della sotto-sequenza, ma lo archivia su database in formato XML, per cui gli attributi e le operazioni della classe sono statici. Ad ogni operazione, il formato XML viene prelevato e convertito in un oggetto ArrayList, sul quale viene svolta l'operazione, e poi archiviato nuovamente in formato XML su database. La Fig.2 mostra (in alto) lo schema del database (da costruire tramite MySQL GUI) e qualche dato di esempio, corrispondente a tre istanze di *PCList*. Nell'esempio le tre istanze risiedono sul medesimo host e ciascuna istanza viene identificata dalla corrispondente porta di ascolto. Le operazioni da svolgere riguardano sempre la sequenza nel suo insieme, e possono essere richieste da qualsiasi oggetto *PCList* (richiedente). Per coordinarsi, gli oggetti *PCList* si scambiano un oggetto *PCStatement* (dal contenuto immutabile durante lo svolgimento di una operazione). Le operazioni possibili sono le seguenti:

- **create**: crea una sequenza fatta da tante sotto-sequenze vuote, in tutti i processi *PCList*. In particolare: il richiedente genera un *ArrayList* vuoto, lo archivia su database in formato XML, lo visualizza, inoltra l'operazione al processo successivo, che fa le medesime azioni, e cosi via fino al processo richiedente, che si limiterà a ricevere l'operazione;
- **add**: inserisce un dato valore nella prima sotto-sequenza di lunghezza minima trovata. In particolare: il richiedente controlla se la propria sotto-sequenza è vuota, nel qual caso inserisce il valore; altrimenti inoltra l'operazione al processo successivo, che fa le medesime azioni, e cosi via. Se nessuno dei processi ha un segmento vuoto, l'operazione ritorna al richiedente. A quel punto si esegue il medesimo protocollo, controllando questa volta se il proprio segmento ha un solo elemento. Se nessuno dei processi ha un segmento di un solo elemento, l'operazione ritorna al richiedente. A quel punto si esegue il medesimo protocollo, controllando questa volta se il proprio segmento ha due soli elementi. Procedendo in questo modo si troverà prima o poi un segmento di dimensione minima dove inserire il nuovo valore. A quel punto l'operazione verrà eseguita, il segmento locale sarà visualizzato, e l'operazione non sarà più propagata;
- **remove**: estrae il valore dal primo segmento che lo contiene, se esiste. In particolare: il richiedente controlla se la propria sotto-sequenza contiene il valore, nel qual caso lo estrae e stampa il contenuto del segmento; altrimenti inoltra l'operazione al processo successivo, che fa le medesime azioni, e cosi via. Se nessuno dei segmenti ha il valore, l'operazione ritorna al richiedente, che si limiterà a ricevere l'operazione.

Si realizzi un'applicazione Java distribuita composta dalle classi *PCList* e *PCStatement*. Le visualizzazioni, gli invii e le ricezioni dell'operazione siano anche in formato XML. **Attenersi esattamente a quanto espresso dalle seguenti figure**. Fig.1 mostra una configurazione di test con soli tre host. Fig.2 mostra i file dell'applicazione e il database. 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. La logica delle due classi dovrà essere valida a prescindere dai numeri di porta e di processi, e dal processo richiedente.

```
@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;%L%mysql-connector-java-5.1.34-bin.jar;

start cmd /k "@echo off && color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "@echo off && color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082"
start cmd /k "@echo off && color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081 create 0"
pause
taskkill /f /im "java.exe"
rem risultato: tutti gli host hanno una lista vuota nel db
pause

start cmd /k "color 3F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "color 3F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082 add 3.14"
start cmd /k "color 3F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081"
pause
taskkill /f /im "java.exe"
rem risultato: lo host 8081 inserisce il dato 3.14
pause

start cmd /k "color 4F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "color 4F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082 add 1.72"
start cmd /k "color 4F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081"
pause
taskkill /f /im "java.exe"
rem risultato: lo host 8082 inserisce il dato 1.72
pause

start cmd /k "color 5F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "color 5F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082 add 5.92"
start cmd /k "color 5F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081"
pause
taskkill /f /im "java.exe"
rem risultato: lo host 8080 inserisce il dato 5.92
pause

start cmd /k "color 6F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "color 6F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082 add 2.16"
start cmd /k "color 6F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081"
pause
taskkill /f /im "java.exe"
rem risultato: lo host 8081 inserisce il dato 2.16
pause

start cmd /k "color 7F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "color 7F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082 remove 5.92"
start cmd /k "color 7F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081"
pause
taskkill /f /im "java.exe"
rem risultato: lo host 8080 rimuove il dato 5.92
pause

start cmd /k "color 8F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8082 8080"
start cmd /k "color 8F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8081 8082 remove 5.9"
start cmd /k "color 8F && c:\prg\jdk8\bin\java -classpath %LIBS% PCList 8080 8081"
pause
taskkill /f /im "java.exe"
rem risultato: nessuno estrae
pause
```

Fig.4 - File make.bat, l'unico ad essere fornito, da non modificare.

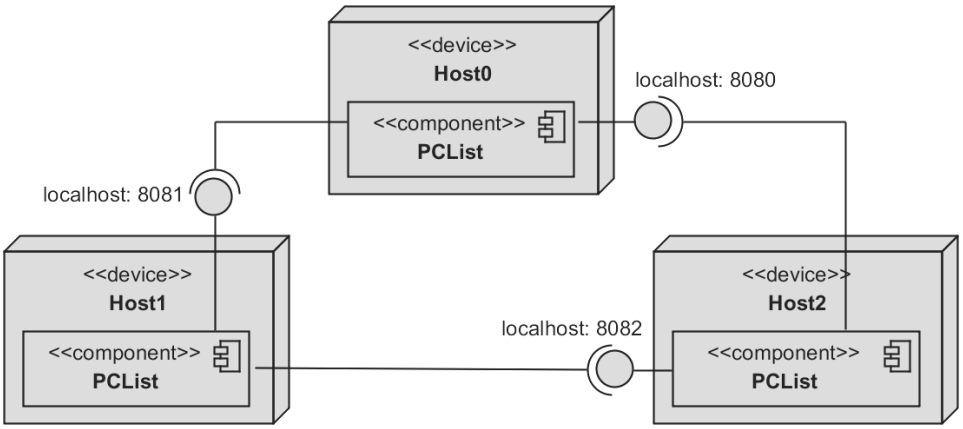

Fig.1 - Tre istanze di *PCList* e relative porte di ascolto

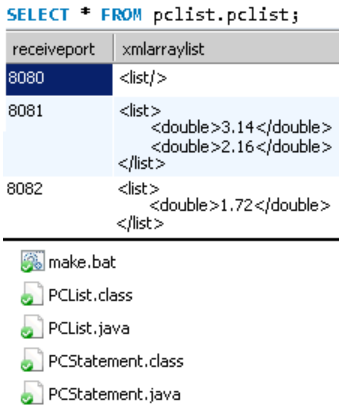

Fig.2 - c:\prg\esercizio2

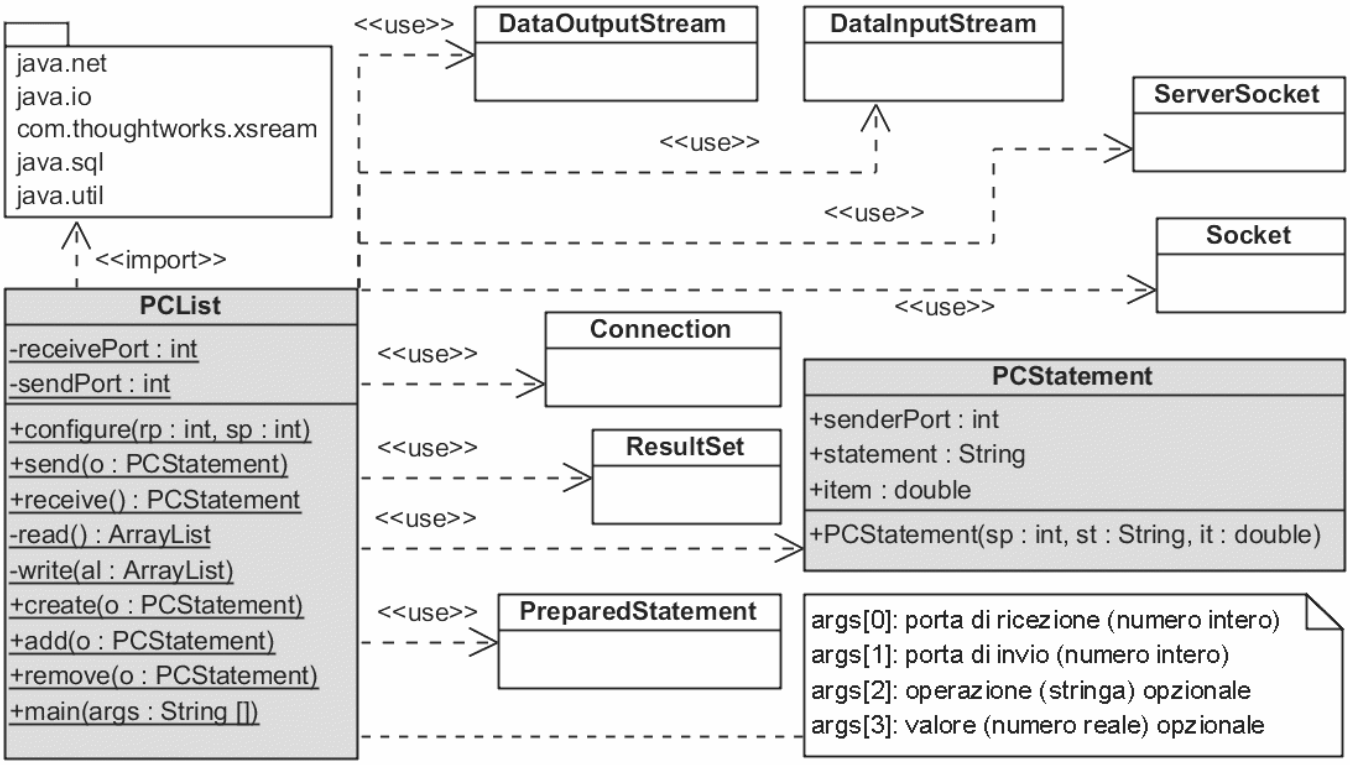

Fig.3 - Classi di cui deve essere composta l'applicazione (in grigio) e classi/package da usare (in bianco)