Programmazione Avanzata 18/02/2022

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

PREMESSA:

Una password in chiaro (P) è una stringa di caratteri. Una password in chiaro può essere data in ingresso a una funzione hash ottenendo la password cifrata. Più precisamente ciò che viene ottenuto è l'hash H della password, che è anch'esso una stringa:

H = hash(P)

La password di un utente viene memorizzata dal sistema operativo in forma cifrata all'interno di un file. In particolare, ciò che viene effettivamente memorizzato è l'hash della password, che chiameremo H1. Quando l'utente esegue il login

- l'utente inserisce una stringa X
- il sistema operativo calcola l'hash di X (chiamiamolo H2)
- il sistema operativo confronta H2 con H1: se sono uguali l'utente è autorizzato, altrimenti no.

Il sistema operativo conserva l'hash della password e non la password vera e propria. In questo modo, se un malintenzionato riuscisse a leggere il contenuto del file non avrebbe in mano la password ma solo il suo hash.

ESERCIZIO: Un malintenzionato ha scoperto l'hash della password di un utente e vuole conoscerne la password in chiaro. Per fare questo adotta un approccio a forza bruta in cui calcola l'hash di un insieme molto grande di stringhe s1, s2, s3, ..., sN. Se una di queste stringhe produce un hash uguale a quello dell'utente allora vuol dire che quella stringa è la password in chiaro dell'utente.

Realizzare una classe *Cracker* dotata almeno dei seguenti metodi e costruttori:

- *Cracker(String h)*: crea un Cracker; *h* è l'hash della password dell'utente di cui il malintenzionato è venuto a conoscenza.
- *void controlla(String[] s, int numThreads)*: controlla se una delle stringhe indicate dal primo argomento è la password in chiaro dell'utente. Per fare questo usa un numero di thread specificato dal secondo argomento. Ogni thread esegue ciclicamente le seguenti azioni: prende una delle stringhe non ancora analizzate, calcola l'hash della stringa, controlla se l'hash appena prodotto è uguale a quello dell'utente.
- *void stampaRisultato()*: stampa a video un messaggio che indica la password in chiaro dell'utente, se è stata trovata. In alternativa, stampa un messaggio in cui si dice che la password non è stata trovata. Il metodo è *bloccante* nel caso in cui venga invocato prima che sia terminata l'analisi delle parole specificate in *controlla()*.

Realizzare anche una classe *Lancia* che può essere usata dal malintenzionato come illustrato dal seguente esempio:

```
java esercizio1.Lancia xItN9WWwyW+E/t8Y8mWW7UCqn0bxECGvcSXTTR06z/4= 4 abc qwerty pluto pippo ingegneria
informatica
Thread-1: controllo qwerty
```

```
Thread-3: controllo pippo
Thread-0: controllo abc
Thread-2: controllo pluto
La password è pluto
```

Il primo argomento (*xItN9...*) è l'hash della password dell'utente, il secondo (*4*) il numero di thread da utilizzare, i rimanenti (*abc qwerty pluto pippo* ...) le parole da testare.
Altro esempio:

```
java esercizio1.Lancia cw912v1z4Ee4asstvXTnXcuTJy+ghKkIKEjyNBqhq7Y= 4 abc qwerty pluto pippo ingegneria
informatica

Thread-0: controllo abc
Thread-1: controllo qwerty
Thread-2: controllo pluto
Thread-3: controllo pippo
Thread-1: controllo informatica
Thread-2: controllo ingegneria
Nessuna delle parole è la password
```

Per calcolare l'hash di una stringa usare il metodo *hash()* della classe *Hash* presente nella cartella *esercizio1*.

Esercizio 2

Un *ArchivioDistribuito* è un documento XML, contenente una sequenza di valori reali, segmentato in vari file XML ciascuno su un diverso host/processo Java. I processi sono comunicanti secondo uno schema ad anello (es. Fig.1), e il loro numero non è noto a priori. Per gestire in modo coordinato i segmenti, si adopera una istanza di *ArchivioDistribuito* per ogni segmento. Ogni oggetto *ArchivioDistribuito* può accedere esclusivamente al proprio file XML. La Fig.2 mostra (in alto) tre esempi di file XML generati da tre distinti processi Java, posti per semplicità sul medesimo host e su porte di ascolto diverse: *8080_archivio.xml*, *8081_archivio.xml*, *8082_archivio.xml*. La Fig.2 mostra (in basso) il contenuto di un file XML. Le operazioni da svolgere riguardano sempre l'archivio nel suo insieme, e possono essere richieste da qualsiasi oggetto *ArchivioDistribuito* (richiedente). Per coordinarsi, gli oggetti *ArchivioDistribuito* si scambiano un oggetto *Operazione* (dal contenuto immutabile durante lo svolgimento di una operazione). Gli accessi al file XML avvengono nel seguente modo: si trasforma il contenuto del file XML in un oggetto temporaneo *ArrayList* e si svolgono le operazioni su tale oggetto; quindi si ritrasforma l'oggetto in XML e si sovrascrive il file. Poichè lo stato dell'archivio viene mantenuto interamente su file, gli attributi e le operazioni della classe *ArchivioDistribuito* sono statici. Le operazioni possibili sono le seguenti:

- **crea**: crea un file XML contenente un segmento senza dati in tutti i processi. In particolare: il richiedente genera un *ArrayList* vuoto, lo archivia su file 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;
- **inserisci**: inserisce un dato valore nel primo file XML di lunghezza minima trovato. In particolare: il richiedente controlla se il proprio segmento è vuoto, 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;
- **estrai**: estrae il valore dal primo segmento che lo contiene, se esiste. In particolare: il richiedente controlla se il proprio segmento 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 *ArchivioDistribuito* e *Operazione*. Le visualizzazioni, gli invii e le ricezioni dell'operazione siano 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. 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.**Commentare ogni riga di codice con una breve spiegazione di cosa fa con riferimento a quanto chiesto dalla traccia e dalle figure.**

```
@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 "@echo off && color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% ArchivioDistribuito 8082 8080"
start cmd /k "@echo off && color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% ArchivioDistribuito 8081 8082"
start cmd /k "@echo off && color 2F && c:\prg\jdk8\bin\java -classpath %LIBS% ArchivioDistribuito 8080 8081 crea 0"
pause
taskkill /f /im "java.exe"
rem risultato: tutti gli host creano un file senza dati
type 8080_archivio.xml
type 8081_archivio.xml
type 8082_archivio.xml
pause

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

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

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

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

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

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

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

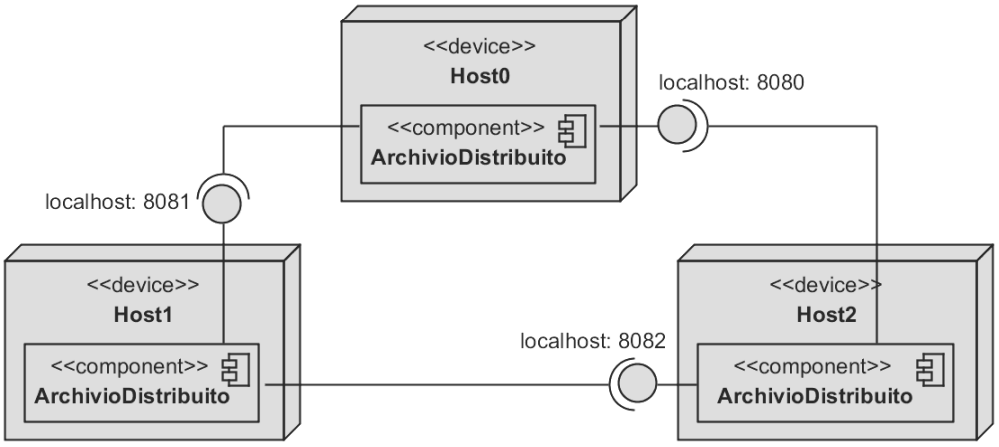

Fig.1 - Tre istanze di *ArchivioDistribuito* e relative porte di ascolto

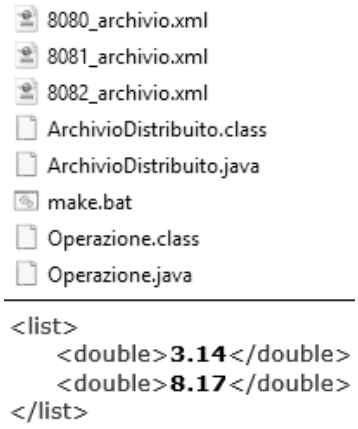

Fig.2 - c:\prg\esercizio2

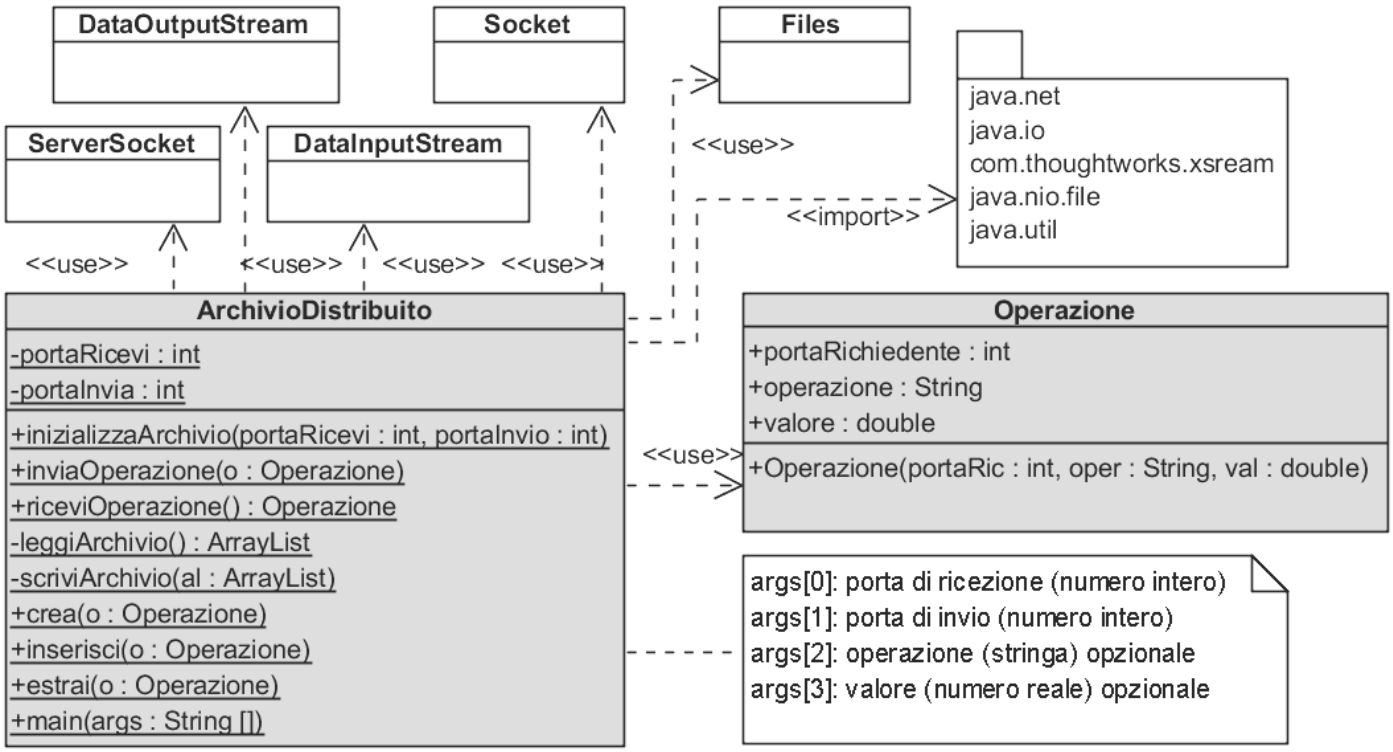

Fig.3 - Classi di cui deve essere composta l'applicazione (in grigio) e classi/package da usare (in bianco)