Programmazione Avanzata 15/01/2020

Note sullo svolgimento della prova

- Books, personal notes, and past exam assignments are not allowed.
- Use NetBeans for editing your source files. You can also use DOS shells for compiling and/or executing your code (in addition to NetBeans)
- The C:\prg folder contains the JDK to be used. The same folder also contains the docs of the Java API. Possible auxiliary files (if any) must be downloaded from elearn.ing.unipi.it.
- The source files of the two exercises must be stored in C:\prg\myapps\esercizio1 and C:\prg\myapps\esercizio2 respectively.
- At the end of the exam, after having uploaded all the files on elearn.ing.unipi.it, the teacher will show you a possible solution.

Exercise 1

A cappucino is made of two ingredients: milk and coffee. There are three baristas and a supplier around a table. Baristas do not have their own supply of ingredients, whereas the supplier is provided with an infinite amount of both coffee and milk. Up to one ingredient can be on the table. Every barista makes infinite cappuccinos by cyclically performing the following actions: i) he waits for the presence of an ingredient on the table and takes the ingredient; ii) he waits for the presence of the *other* ingredient on the table and takes the ingredient. Every time an ingredient is taken from the table, the table becomes empty. The supplier behaves cyclically as follows: i) he randomly chooses one of the two ingredients (coffee or milk); ii) he waits for the table to be empty and then puts the chosen ingredient onto the table. If an ingredient remains on the table for more than 3 seconds, the supplier removes the ingredient currently available on the table and changes the ingredient with the other one (if, for instance, coffee remains on the table for more than 3 seconds, the supplier removes coffee and puts milk onto the table).
You have to implement the Table, Barista, and Supplier classes. Busy waiting must be avoided.
Example of possible execution:

```
Supplier: I put MILK
Supplier: the table is not empty, blocking.
Barista 1: I took MILK
Supplier: I put MILK
Supplier: the table is not empty, blocking.
Barista 3: I took MILK
Supplier: I put MILK
Supplier: the table is not empty, blocking.
Barista 2: I took MILK
Supplier: I put MILK
Supplier: the table is not empty, blocking.
Supplier: timeout expired, removing the ingredient (MILK)
Supplier: changing ingredient.
Supplier: I put COFFEE
Supplier: the table is not empty, blocking.
Barista 2: I took COFFEE
Supplier: I put COFFEE
Supplier: the table is not empty, blocking.
Barista 3: I took COFFEE
Supplier: I put COFFEE
Supplier: the table is not empty, blocking.
Barista 1: I took COFFEE
Supplier: I put COFFEE
Supplier: the table is not empty, blocking.
Barista 2: I took MILK e poi COFFEE: cappuccino!!!
Barista 2: I took COFFEE
Supplier: I put MILK
Supplier: the table is not empty, blocking.
Barista 1: I took MILK e poi COFFEE: cappuccino!!!
...
```

Exercise 2 (generated with Google Translator)

The *PCList (Persistent Circularly Linked List)* class manages a sequence of real values, divided into sub-sequences on a non-predefined number of communicating hosts according to a ring scheme (eg Fig.1). To manage the sub-sequences in a coordinated way, an instance of PCList is used for each of them. The PCList class does not maintain the state of the sub-sequence, but stores it on databases in XML format, so the attributes and operations of the class are static. At each operation, the XML format is taken and converted into an ArrayList object, on which the operation is carried out, and then archived again in XML format on a database. Fig.2 shows (at the top) the database schema (to be built through MySQL GUI) and some example data, corresponding to three instances of PCList. In the example, the three instances reside on the same host and each instance is identified by the corresponding listening port. The operations to be carried out always concern the sequence as a whole, and can be requested by any PCList object (initiator). To coordinate, PCList objects exchange a PCStatement object (with unchangeable content during the course of an operation). The possible operations are the following:

● **create**: creates a sequence made up of many empty sub-sequences, in all PCList processes. In particular: the initiator generates an empty ArrayList, stores it on database in XML format, displays it, forwards the operation to the next process, which does the same actions, and so on until the requesting process, which will simply receive the operation;

● **add**: inserts a given value in the first sub-sequence of minimum length found. In particular: the initiator checks whether its sub-sequence is empty, in which case it inserts the value; otherwise, it forwards the operation to the next process, which does the same actions, and so on. If none of the processes has an empty segment, the operation returns to the initiator. At that point the same protocol is carried out again, checking if the segment has only one element. If none of the processes has one segment, the operation returns to the initiator. At that point the same protocol is carried out again, checking if the segment has two elements. At a certain point a segment of minimum size is found, where to insert the new value. Then the operation is performed, the local segment is displayed, and the operation will no longer be propagated;

● **remove**: extracts the value from the first segment that contains it, if it exists. In particular: the initiator checks whether its sub-sequence contains the value, in which case it extracts it and shows the segment content; otherwise, it forwards the operation to the next process, which does the same actions, and so on. If none of the segments has the value, the operation returns to the initiator, which simply receives the transaction.

Create a distributed Java application consisting of the *PCList* and *PCStatement* classes. Any visualization, send and receive of the operation are also in XML format. **Follow exactly what is expressed in the following figures**. Fig. 1 shows a test configuration with only three hosts. Fig.2 shows the application files and the database. Fig.3 shows the classes to be implemented (in gray), with the related attributes and methods to be created, the packages and classes to be imported and used. Manage the closure of open flows and connections. Capture only the mandatory exceptions, and manage them simply by printing related information. Fig. 4 shows the make.bat testing file, with some test cases. The logic of the two classes must be valid regardless of the port, the process numbers, and the requesting process.

```
@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 result: all hosts have an empty list in their 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 result: host 8081 inserts the value 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 result: host 8082 inserts the value 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 result: host 8080 inserts the value 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 result: host 8081 inserts the value 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 result: host 8080 removes the value 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: no removal
pause
```

Fig.4 - make.bat file, already included, not to be changed.

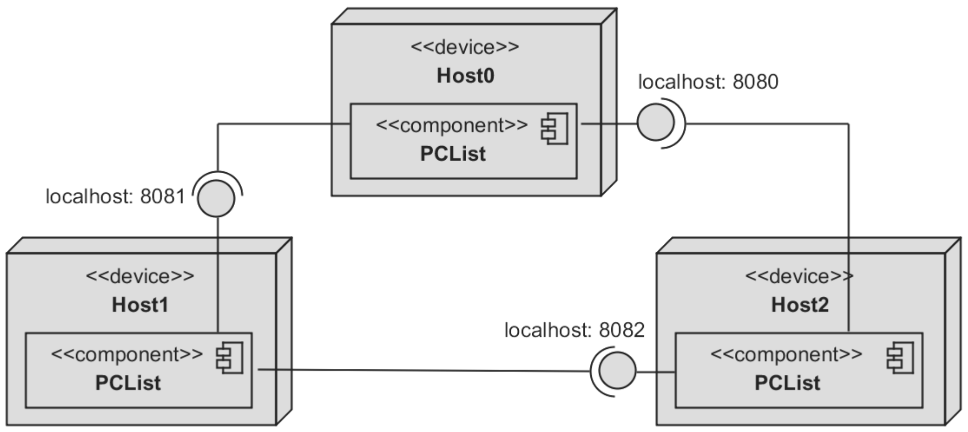
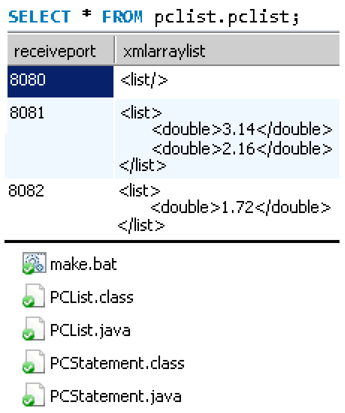


Fig.1 - Three instances of PCList and the related listening ports



```
SELECT * FROM pclist.pclist;
```

receiveport	xmlarraylist
8080	`<list/>`
8081	`<list>` `<double>3.14</double>` `<double>2.16</double>` `</list>`
8082	`<list>` `<double>1.72</double>` `</list>`

- make.bat
- PCList.class
- PCList.java
- PCStatement.class
- PCStatement.java

Fig.2 - c:\prg\esercizio2

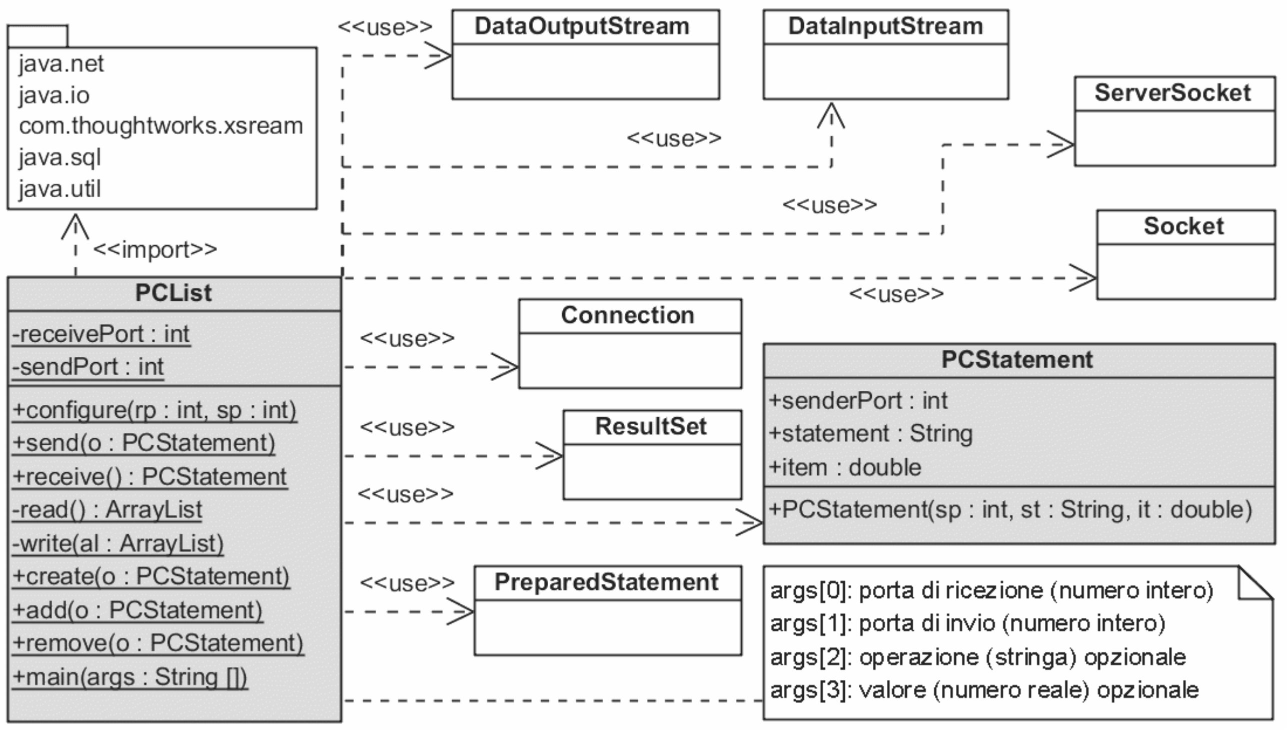


Fig.3 - Classes to be implemented (in gray), packages and classes to be imported and used (white).