Programmazione Concorrente e Distribuita

DOMANDE TEORICHE

1. Nel pattern Observer quale affermazione è vera?
   Gli observer mettono a disposizione dell'oggetto osservato un metodo per comunicare notifiche
2. Quale è la funzione dei Registry in RMI?
   Fornire al client il riferimento al server
3. Qual è l'effetto di una sincronizzazione di tipo Barrier quando la barriera è inizializzata con parametro uguale a 10 (new Barrier(10))?
   I task in attesa alla barriera sono sbloccati alla decima chiamata del emtodo waitB();
4. Quale delle seguenti condizioni deve essere soddisfatta per creare un server capace di gestire più connessioni contemporaneamente?
   Far gestire ogni socket creato da ServerSocket.accept() ad un nuovo thread.
5. Quale delle seguenti affermazioni riguardanti un oggetto proxy è vera?
   La funzione principale del proxy è sollevare un client dalla gestione delle comunicazioni
6. Quale tra I seguenti problemi affronta il pattern produttore/consumatore?
   Assicura che il produttore non cerchi di inserire mai dati se il buffer è pieno, e che consumatore non estragga dati se il buffer è vuoto
7. A cosa serve lo scheduler della JVM?
   Stabilisce quale thread mandare in esecuzione selezionandolo dal ready set
8. Quale delle seguenti affermazioni è vera, relativamente a callback basate su RMI?
   Il server chiama un metodo dal client, di cui possiede il riferimento remoto
9. Quando un thread esegue notify() che evento si verifica?
   Viene sbloccato un thread fra quelli in wait
10. Cosa fa l'istruzione Thread.sleep(3000)?
    Addormenta il currentThread per 3000ms
11. Quale delle seguenti condizioni deve essere verificata perchè un client remoto possa inviare un messaggio a un server remoto?
    Il client ha ottenuto il riferimento remoto del server dal registry o altrimenti.
12. Qualce condizione deve essere soddisfatta per implementare in Java il problema di produttore/consumatore?
    Gli oggetti produttore/consumatore devono essere thread
13. Qual è l'effetto della barrier sulla sincronizzazione tra thread?
    Bloccare I thread alla barriera fino a quando non sono giunti tutti, poi sbloccarli tutti insieme
14. Cosa server per identificare un server in modo univoco?
    Indirizzo IP e numero porta del server
15. Quale affermazione riguardante uno skeleto è vera?
    La funzione dello skeleton è sollevare il server dalla gestione delle comunicazioni

16. Qual è l’effetto di T.join()?
    Attendere la terminazione del thread T

17. Qual è vera, riguardo gli argomenti dei metodi remoti in RMI?
    Gli argomenti di tipo primitivo non sono serializzabili

18. Guardando questi due blocchi:
    synchronized(this) { T.sleep(1000) }
    synchronized(this) { wait() }
    Cosa succede?
    T.sleep non rilascia il lock in suo possesso mentre wait lo rilascia sull’oggetto this

19. Che protocollo viene usato da serverSocket per far comunicare server e client?
    TCP

20. Cosa si intende con callback?
    Il fatto che il server chiama metodi dal client

21. Sapendo che un programma Java è in esecuzione su un solo processore e contiene n thread nello stato ready, come vengono eseguiti?
    E’ impossibile stabilire la sequenza di esecuzione dagli n thread

22. Si consideri il seguente codice:
    ```
    public void inc1(){
        c1 += c2;
    }
    ```
    Quanti thread possono eseguire contemporaneamente l’istruzione c1 += c2?
    Un numero indefinito

PROCEDURA PER CONCORRENZA

1. Mettere **synchronized** dove necessario ai metodi
2. Di solito si toglie l'if e si sostituisce con un while mettendo all'interno un **wait()**.
3. Alla fine in base all'esercizio si mette un **notifyAll()**;
4. Se l'esercizio richiede che tutti i thread eseguano una esecuzione a testa e si fermino allora si può usare il **CyclicBarrier**

interface non va modificato

nella classe con MAIN mettere extends thread
- creo costruttore
- creo metodo mySleep //dovrebbe essere sempre cosí

```
private void mySleep(int i1, int i2) {
try {
      Thread.sleep(ThreadLocalRandom.current().nextInt(i1, i2));
} catch (InterruptedException e) { }
}
```

- creare metodo run()
- ovviamente modifico il main mettendo tutto quello che non deve esserci nel run()

classe SENZA main:
- lascio costruttore come lo trovo
- metto synchronized in tutti i metodi necessari

Si modifichi la funzione di ricerca in modo che se un thread cerca l'informazione corrispondente ad una chiave non presente, si metta in attesa che qualche altro thread aggiunga il dato cercato.

```
public String trovaDato(String key){
  return iDati.get(key);
}
```

TRASFORMATO

```
public synchronized String trovaDato(String key){
  System.out.println("Deposito dati: "+Thread.currentThread().getName()+" cerca info di " + key);
  while(!iDati.containsKey(key)) {
      System.out.println("Deposito dati: "+Thread.currentThread().getName()+" va in attesa");
      try {
          wait();
  } catch (InterruptedException e) { }
  }
  String info=iDati.get(key);
  System.out.println("Deposito dati: trovato "+info+" di " + key);
  return info;
}
```

PROCEDURA PER SOCKET

1. Eliminare il main dell'applicazione concorrente
2. Serializzare la classe risorsa nel seguente modo:
   - implements Serializable
   - public static final long SerialVersionUID = 1L;
3. I thread sono client:
   - Tolgo extends Thread
   - Tolgo start()
   - Tolgo tutto quello che riguarda la classe condivisa nel costruttore
   - Nel metodo **run** scrivo:
     - InetAddress addr = InetAddress.getByName(null);
     - Socket clientSocket = new Socket(addr, 8080);
     - ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());
     - ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
     - writeObject serve per inviare un oggetto. ReadObject server per ricevere un oggetto.
     -
     - out.writeObject("Stringa da passare"); (RICORDA: passare anche l'argomento!)
     - in.readObject() ricordare di fare il cast con il tipo o oggetto utilizzato.
     - Alla fine di tutto fare:
       - out.writeObject("END");
       - out.flush();
       - ClientSocket.close();
   - Si crea il **main** della classe:
     - new nomeclasse(999); oppure fare Random.
4. Creo classe ServerMain:

```
1  package Es2;
2
3  import java.io.IOException;
6
7  public class ServerMain {
8      public static final int PORT = 8080;
9      public static int numGiocatori = 4;
10
11     public static void main(String[] args) throws IOException {
12         ServerSocket Server = new ServerSocket(PORT);
13         Partita laPartita = new Partita(numGiocatori);
14
15         try {
16             while(true) {
17                 Socket ClientSocket = Server.accept();
18                 new ServerThread(ClientSocket, laPartita);
19             }
20         } finally {
21             Server.close();
22         }
23     }
24 }
```

5. Creo classe ServerThread:
   - Come prima cosa metto extends Thread.
   - Dichiaro ObjectInputStream e ObjectOutputStream
   - Dichiaro la classe condivisa
   - Dichiaro: Socket clientSocket;

- Dichiaro il costruttore nel seguente modo:
  - public ServerThread(Socket s, classeCondivisa oggettoCondiviso){
    - faccio tutte le assegnazioni del caso ed istanzio ObjectInput e Output Stream
    - Alla fine faccio: **start();**
- Creo:

```
21    private void exec(String str) throws IOException, ClassNotFoundException {
22        int id = 0;
23        if(str == "aspettaTurno") {
24            id = (int)in.readObject();
25            laPartita.aspettaTurno(id);
26            out.writeObject(laPartita.leggiSituazione());
27        }
28        if(str == "giocata") {
29            int mossa = (int)in.readObject();
30            laPartita.giocata(id, mossa);
31        }
32        if(str == "numMani") {
33            out.writeObject(laPartita.numMani);
34        }
35    }
36
```

```
37    public void run() {
38        boolean finito = false;
39        String str = "";
40
41        try {
42            while(!finito) {
43                str = (String)in.readObject();
44                if(str == "END") {
45                    finito = true;
46                }else {
47                    System.out.println("Slave esegue: " + str);
48                    exec(str);
49                }
50            }
51            System.out.println("Closing...");
52        } catch (Exception e) {
53
54        } finally {
55            try {
56                ClientSocket.close();
57            } catch (IOException e) {
58                e.printStackTrace();
59            }
60        }
61
62    }
63 }
```

PROCEDURA PER RMI

1. Copiare codice da Socket
2. Cancellare ServerMain e ServerThread.
3. Nel Client:
    1. Togliere tutto quello che riguarda i socket
    2. Registry registro = LocateRegistry.getRegistry(1099);
    3. ServerInterface serverInt;
    4. serverInt = (ServerInterface) registro.lookup(“nome”)
    5. Creare variabile oggettoCondiviso.
4. Creare ServerInterface
    1. Prendere tutti i metodi della classe condivisa
    2. Togliere tutti i synchronized
    3. Aggiungere a tutti I metodi: **throws RemoteException**
    4. **IMPORTANTE**: tutti i metodi devono essere public
5. Creare ServerImpl
    1. extends UnicastRemoteObject implements ServerInterface
        1. public ServerImpl() throws RemoteException {
        2. super();
        3. oggettoCondiviso = new classeCondivisa();
        4. }
    2. Si copiano tutti I metodi che si trovano dentro ServerInterface e si fa il return nel caso fossero diversi da void.
    3. Nel main:

```
public static void main(String[] args) throws RemoteException {
    Registry registro = LocateRegistry.createRegistry(1099);
    ServerImpl serverImpl = new ServerImpl();
    registro.rebind("ServerPartita", serverImpl);
}
```