



## Esame di Ingegneria del Software

Corso 085885  
Prof. Giovanni Ennio Quattrocchi

08-07  
Data: 17-06-2024

Nome e Cognome: .....  
Matricola: .....

**Durata dell'esame: 2 ore**

*Punteggio massimo: 33 punti. Gli studenti possono scrivere in penna o in matita per rispondere alle domande. NON è consentito consultare il materiale didattico.*

### Esercizio 1 – Multithreading (13 punti)

Il calcolo della media mobile di un insieme di dati può essere ottimizzato utilizzando il multithreading. Si richiede l'implementazione di una classe Java che calcola la media mobile di una lista di numeri interi utilizzando il multithreading, per sfruttare il parallelismo e migliorare le prestazioni su grandi insiemi di dati. Si crei una classe Java denominata **MovingAverageCalculator** che offre un metodo statico **calculateParallelMovingAverage** il quale calcola e restituisce la media mobile di un insieme di dati numerici.

Il metodo **calculateParallelMovingAverage** prende in input una lista di numeri interi **data**, un intero **windowSize** e restituisce una lista di double che rappresenta la media mobile dei numeri nella lista calcolati su chunk di dati grandi **windowSize**. Il metodo deve dividere la lista di input in sottoinsiemi di dimensioni adeguate e assegnare a ogni thread il compito di calcolare la media mobile per il suo sottoinsieme.

Ad esempio, data la lista [1, 2, 3, 4, 5, 6, 7] e una **windowSize=4** il metodo restituisce

[avg(1,2,3,4), avg(2,3,4,5), avg(3,4,5,6), avg(4,5,6,7)] ovvero [2.5, 3.5, 4.5, 5.5]

### Esercizio 2 – Testing (4 punti)

Si consideri il seguente frammento di codice Java:

```
public Integer calculateShippingCost(int weight, int distance, boolean isExpress) {  
    int cost = 0;  
  
    if (weight > 10) {  
        cost += 15;  
    }  
  
    if (distance > 100) {  
        cost += 20;  
    }  
  
    if (isExpress) {  
        cost += 30;  
    } else {  
        cost += 5;  
    }  
  
    return cost;  
}
```

② 3 condizioni → 2<sup>3</sup> percorsi

<u>weight</u>	<u>distance</u>	<u>isExpress</u>	RISULTATO (cost)
5	50	X	0
15	50	X	15
5	150	X	20
5	50	✓	5
15	150	✓	40
15	150	X	35
15	50	✓	20
5	150	✓	25

- a) Si definisca un insieme minimo di test utilizzando il criterio di copertura dei cammini.

### Esercizio 3 - UML (12 punti)

DECISION STATEMENT

Implementare un sistema di prenotazione eventi che permetta agli utenti di selezionare tra una vasta gamma di opzioni di partecipazione, tra cui conferenze, concerti, workshop, o festival. Ogni prenotazione ha uno stato, che può essere "pagato e confermato", "pagato e non confermato", "non confermato", "da pagare e confermare" o altri stati che possono essere definiti in futuro. Gli utenti possono aggiungere servizi aggiuntivi e cumulabili alle prenotazioni, come servizio di catering, accesso VIP, parcheggio riservato, workshop extra, o merchandising. Gli amministratori di sistema possono eseguire azioni sulle prenotazioni ad un particolare evento. Tali azioni possono essere: "spedisci biglietti", "annulla", "conferma" e altre definibili in future versioni.


STATO

Si definisca un Class Diagram, utilizzando opportuni design pattern, che descriva le entità rilevanti del dominio applicativo, indicando metodi e attributi significativi e le associazioni tra le entità. Si assegni un nome a ciascuna associazione e si forniscano indicazioni sui vincoli di cardinalità. Si commentino le scelte effettuate quando necessario.

### Esercizio 4 – Generics (4 punti)

Si vuole implementare un metodo generico in Java che prenda due liste di elementi di tipo qualsiasi T e ritorni una nuova lista contenente gli elementi comuni alle due liste.

```
public static <T> List<T> genericMethod(List<T> list1, List<T> list2) {  
    List<T> returnList = new ArrayList<>();  
    Set<T> set2 = new HashSet<>(list2); → conversione list2 da List a Set:  
    for(T element : list1) { → contains() → O(n) vs O(1)  
        if(set2.contains(element)) {  
            returnList.add(element);  
        }  
    }  
    return returnList;  
}
```



```

public class MovingAverageCalculator {

    public static double[] calculateParallelMovingAverage(double[] data, int
windowSize) throws InterruptedException {
        if(windowSize >= data.length){
            windowSize = data.length;
        }

        Object lock = new Object();
        double[] result = new double[data.length - windowSize + 1];

        List<Thread> threads = new ArrayList<>();
        for(int i = 0; i < data.length; i++){
            final int startIndex = i;
            final int endIndex = startIndex + windowSize;
            if(endIndex <= data.length){
                int finalWindowSize = windowSize;
                Thread thread = new Thread(() ->{
                    synchronized(lock){
                        double average = 0;
                        for(int j = startIndex; j < endIndex; j++){
                            average += data[j];
                        }
                        average /= finalWindowSize;
                        result[startIndex] = average;
                    }
                });
                thread.start();
                threads.add(thread);
            }
        }

        for (Thread thread : threads) {
            thread.join();
        }

        return result;
    }
}

public static void main(String[] args) {
    double[] data = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    int windowSize = 3;

    try {
        // Chiamo il metodo per calcolare la media mobile in parallelo
        double[] movingAverages =
MovingAverageCalculator.calculateParallelMovingAverage(data, windowSize);

        // Stampo i risultati
        System.out.println("Risultati delle medie mobili:");
        for (double avg : movingAverages) {
            System.out.println(avg);
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

Risultato: 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0