



Esame di Ingegneria del Software

Corso 085885
Prof. Giovanni Ennio Quattrocchi

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 (15 punti)

Il calcolo della varianza di un insieme di dati è un'operazione comune nell'analisi statistica. Si richiede l'implementazione di una classe Java che calcola la varianza 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 **VarianceCalculator** che offre un metodo statico **calculateParallelVariance** il quale calcola e restituisce la varianza di un insieme di dati numerici.

Il metodo **calculateParallelVariance** prende in input una lista di numeri interi **data**, un intero **subIntervals**, e restituisce un valore double che rappresenta la varianza dei numeri nella lista.

Il metodo include due fasi: la prima calcola la media della lista, nella seconda viene calcolata la varianza. In entrambe le fasi il metodo deve dividere la lista di input in una quantità pari a **subIntervals** di sottoinsiemi e assegnare a ogni thread il compito di calcolare un risultato parziale.

Si ricorda che la varianza di un insieme si calcola con la seguente formula:

$$\sigma_X^2 = \frac{\sum_i (x_i - \mu_X)^2}{n}$$

Dove X è l'insieme numerico, σ_X^2 è la varianza di X , x_i è l'elemento i -esimo dell'insieme, μ_X è la media di X , e n è il numero di elementi incluso in X .

Esercizio 2 – Testing (3 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 += 5;  
    }  
  
    return cost;  
}
```

	weight	distance	isExpress	cost
①	11	11	T	65
②	0	0	F	5

```

    cost += 30;
} else {
    cost += 5;
}

return cost;
}

```

a) Si definisca un insieme minimo di test utilizzando il criterio di copertura delle istruzioni.

Esercizio 3 - UML (12 punti)

Implementare un sistema di prenotazione viaggi che permetta agli utenti di selezionare tra una vasta gamma di opzioni di alloggio, tra cui hotel, bed and breakfast, appartamenti o ostelli.

Il sistema dovrebbe supportare i seguenti requisiti:

- Ogni prenotazione ha uno stato, che può essere "in attesa di conferma", "cancellabile", "senza cancellazione" o altri stati che possono essere definiti in futuro.
- Gli utenti possono aggiungere servizi aggiuntivi e cumulabili alle prenotazioni, come servizio transfer dall'aeroporto, cancellazione gratuita, disponibilità parcheggio, colazione inclusa o visite guidate per rendere il tuo viaggio ancora più confortevole e completo.
- Gli utenti possono modificare le impostazioni per ricevere notifiche, ad esempio per ricordare le scadenze del periodo di cancellazione gratuita o se avvicina una prenotazione.
- Gli utenti possono cancellare le prenotazioni, categorizzarle e valutarle successivamente.

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 assegna un nome a ciascuna associazione e si forniscano indicazioni sui vincoli di cardinalità. Si commentino le scelte effettuate quando necessario.

Esercizio 4 – Generics (3 punti)

Si vuole implementare una classe generica in Java denominata **Pair** che rappresenta una coppia di oggetti di qualsiasi tipo. La classe deve avere i seguenti requisiti:

- Due variabili di istanza, **first** e **second**, di tipo generico **T**.
- Un costruttore che accetta due parametri di tipo **T** e li assegna alle variabili di istanza.
- Metodi getter per ottenere i valori di **first** e **second**.
- Metodi setter per impostare i valori di **first** e **second**.
- Un metodo **swap** che scambia i valori di **first** e **second**.

- a) Si definisca la classe generica **Pair**
 b) Si crei un main che utilizzi la classe **Pair** con diversi tipi di dati

```
public class Pair{}
```

```
private T first
```

```
private T second
```

```
public Pair (T first, T second) {
```

```
    this.first = first
```

```
    this.second = second
```

```
}
```

```
public void setFirst (T first) { this.first = first }
```

```
public void setSecond (T second) { this.second = second }
```

```
public T getFirst () { return first }
```

```
public T getSecond () { return second }
```

```
public void swap () {
```

```
    T temp = first
```

```
    first = second
```

```
    second = temp
```

```
}
```

```
}
```

MAIN:

```
Pair<String> pairString = new Pair<>("Hello", "World");
```

```
pairString.swap()
```

```
System.out.println("First word: " + pairString.getFirst() + ", second word: " +
```

```
pairString.getSecond());
```

```
Pair<String> incompatiblePair = new Pair<>("Hello", 123); → RUNTIME ERROR
```

// Esempio di casting per ammorcire il tipo corretto

```
Pair<generic> genericPair = new Pair<>(123, 456);
```

```
Pair<Integer> intPair = new (Pair<Integer>) genericPair; → cast non produce  
errore
```

