



POLITECNICO  
MILANO 1863

## Esame di Ingegneria del Software

Corso 085885  
Prof. Giovanni Ennio Quattrocchi

Data: 20-06-2022

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 (12 punti)

La classe SciCommons espone un metodo statico sample che prende in input una matrice di numeri interi di grandezza (N,N) di tipo Integer[][], e che produce in output un vettore Integer[] di lunghezza N\*2+2 contenente per ogni riga, colonna e diagonale un numero casuale estratto tra quelli contenuti nella rispettiva riga, colonna, e diagonale.

Ad esempio data in un input la matrice:

1 2 3  
4 5 6  
7 8 9

un possibile risultato è:

1, 5, 7, 4, 5, 9, 5, 3

Il metodo deve per prima cosa controllare che la matrice sia quadrata. Per velocizzare il calcolo, il metodo sample è implementato con il multithreading utilizzando N\*2+2 thread. Ciascun thread deve estrarre casualmente un valore data una data riga, colonna, o diagonale. Quando tutti i thread hanno terminato, il metodo compone il vettore risultante e ne ritorna il valore.

### Esercizio 2 – Testing (5 punti)

Si consideri il seguente frammento di codice, supponendo che tutte le variabili usate siano di tipo intero o array di interi:

```
1 L = [45, 0, 3, 0, 4]
2 a = 0
3 b = 1
4 if (x > 0)
5   a -= 1 ①
6 else
7   a += 1 ②
8 if (y > 0)
9   a += 1 ③
10 if (y > 10)
11   b += 2 ④
```

	x	y	a	b	RISULTATO
①	100	-5	-1	2	L(-1) → out of bound
②	-5	100	2	3	3/0 → errore

```
12 else
13     b += 1 ①
14 return L[a]/L[b]
```

- a) (1 punto) Si definisca il diagramma di flusso relativo al codice riportato.  
b) (4 punti) Si definisca un insieme minimo di test seguendo il principio della copertura delle istruzioni, riportando il risultato o errore atteso

### Esercizio 3 - UML (11 punti)

Si descrivono i principi dei design pattern **Command** e **State**. Per ciascuno di questi si faccia un esempio di applicazione con relativo class diagram. Non è ammesso riportare gli esempi presenti nel materiale didattico (video lezioni ed esercitazioni).

### **Esercizio 4 – Generics (5 punti)**

Si implementi una classe **Triple** parametrica rispetto a tre tipi. La classe rappresenta una struttura dati che contiene 3 elementi ordinati. Ad esempio:

```
Triple<String, Integer, Float> t = new Triple<>("Hello", 99, 4.0)
```

La classe espone i metodi getter (**getFirst**, **getSecond**, **getThird**) e setter (**setFirst**, **setSecond**, **setThird**) per ottenere e modificare i valori del primo, secondo o terzo elemento.

Ad esempio:

```
t.getSecond() // 99
t.setThird(5.0)
t.getThird() // 5.0
```

Inoltre, il metodo **reverse** permette di ottenere una nuova istanza di **Triple** contenente gli stessi elementi di quella originale ma in ordine inverso.

Ad esempio:

```
Triple<Float, Integer, String > t2 = t.reverse()
t2.getThird() // "Hello"
```

06/2022

public class Triple { < F, S, T > }

private F first;

private S second;

private T third;

CONSTRUCTOR:

public Triple (F first, S second, T third) {

this.first = first

this.second = second

this.third = third

}

GETTERS:

public F getFirst() {

return first

}

public S getSecond() {

return second

}

public T getThird() {

return third

}

SETTERS:

public void setFirst (F first) {

this.first = first

}

public void setSecond (S second) {

this.second = second

}

public void setThird (T third) {

this.third = third

}

public Triple < ~~F, S, T~~, <sup>T, S, F</sup> > reverse() {

Triple < T, S, F > returnTriple = new Triple <>();

returnTriple.setFirst (this.getThird())

returnTriple.setSecond (this.second)

returnTriple.setThird (this.first)

return returnTriple

}

~~OPPURE~~

**OPPURE**

~~Return~~ Triple < T, S, F > returnTriple = new Triple <> (this.third, this.second, this.first)

return returnTriple

**OPPURE**

return new Triple <> (this.third, this.second, this.first)

```

public static Integer[] sample(Integer[][][] matrix) throws InterruptedException{
    int matrixLength = matrix.length;

    // Check if matrix is square
    for(int i = 0; i < matrix.length; i++){
        if(matrix[i].length != matrixLength){
            throw new IllegalArgumentException("Matrix is not square");
        }
    }

    // Initialize variables
    int returnLength = matrixLength * 2 + 2;
    Integer[] returnList = new Integer[returnLength]; //returnList structure:
    columns (left->right), rows (top->bottom), diagonals (1. left-right, 2. right-left)
    Thread[] threads = new Thread[returnLength];
    Random random = new Random(); //For random indexes

    //*** Columns ***
    for(int i=0; i < matrixLength; i++){
        final int columnIndex = i; //final to ensure the variable will not be
        modified after initialization (is not mandatory but ensures a correct variable usage
        threads[columnIndex] = new Thread(new Runnable() {
            @Override
            public void run(){
                int randomIndex = random.nextInt(matrixLength);
                returnList[columnIndex] = matrix[randomIndex][columnIndex];
            }
        });
        SI POTREBBE SCRIVERE COME:
        threads[columnIndex] = new Thread(() -> {
            int randomIndex = random.nextInt(matrixLength);
            returnList[columnIndex] = matrix[randomIndex][columnIndex];
        });
        threads[columnIndex].start();
    }

    //*** Rows ***
    for(int i=0; i < matrixLength; i++){
        final int rowIndex = i;
        threads[rowIndex + matrixLength] = new Thread(() -> {
            int randomIndex = random.nextInt(matrixLength);
            returnList[rowIndex + matrixLength] = matrix[rowIndex][randomIndex];
        });
        threads[rowIndex + matrixLength].start();
    }

    //*** Left-right diagonal ***
    threads[matrixLength * 2] = new Thread(() -> {
        int randomIndex = random.nextInt(matrixLength);
        returnList[matrixLength * 2] = matrix[randomIndex][randomIndex];
    });
    threads[matrixLength * 2].start();

    //*** Right-left diagonal ***
    threads[matrixLength * 2 + 1] = new Thread(() -> {
        int randomIndex = random.nextInt(matrixLength);
        returnList[matrixLength * 2 + 1] = matrix[randomIndex][matrixLength - 1 -
randomIndex];
    });
    threads[matrixLength * 2 + 1].start();

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

    return returnList;
}

```