



# Esame di Ingegneria del Software

**POLITECNICO**  
MILANO 1863

Corso 085885  
Prof. Giovanni Ennio Quattrocchi  
Data: 18-01-2021

**Durata dell'esame: 2 ore (Online)**

*Punteggio massimo: 33 punti. Gli studenti possono scrivere in penna o in matita per rispondere alle domande e possono consultare il materiale didattico in forma digitale.*

## Esercizio 1 – Multithreading (11 punti)

Si consideri la classe **MessageQueue** riportata di seguito.

```
class MessageQueue {  
    private List<String> data;  
    private int size;  
  
    public MessageBuffer(int size) {  
        data = new ArrayList<>();  
        this.size = size;  
    }  
  
    public ... enqueue(...) {  
        ...  
    }  
    public ... dequeue(...) {  
        ...  
    }  
}
```

**MessageQueue** implementa una coda FIFO (First In First Out) di messaggi (oggetti di tipo **String**). I messaggi vengono salvati nella lista **data** di dimensione massima **size**.

a) (6 punti) Si completi l'implementazione della classe **MessageQueue** in modo che questa sia utilizzabile da più thread. In particolare:

- Definire ed implementare il metodo **enqueue** in modo tale che thread paralleli possano sequenzialmente aggiungere un messaggio in coda alla lista **data**. Se il numero di elementi in **data** è uguale a **size**, allora il thread in esecuzione deve fermarsi ed attendere finché la lista **data** non è più piena.
- Definire ed implementare il metodo **dequeue** in modo tale che thread paralleli possano sequenzialmente rimuovere messaggi dalla lista **data**. I messaggi vanno rimossi in ordine di aggiunta (FIFO) ovvero viene rimosso per primo il messaggio aggiunto prima di tutti gli altri. Se il numero di elementi in **data** è uguale a 0, allora il thread in esecuzione deve fermarsi ed attendere finché la lista **data** non è più vuota.

```
FIFO queue of messages (type is String)
Messages are saved in data List of max size size
public class MessageQueue {
    private List<String> data;
    private int size;

    public MessageQueue(int size) {
        data = new ArrayList<>();
        this.size = size;
    }

    public void enqueue(String message) throws InterruptedException {
        synchronized(this){
            while(data.size() >= size){
                // CODA PIENA
                System.out.println("Full queue...");
                wait();
            }
            // CODA NON PIENA
            data.add(message);
            System.out.println("Added to queue: " + message);
            notifyAll();
        }
    }

    public String dequeue() throws InterruptedException {
        synchronized(this){
            while(data.isEmpty()){
                // CODA VUOTA
                System.out.println("Empty queue...");
                wait();
            }
            // CODA NON VUOTA
            String removedElement = data.remove(0);
            System.out.println("Removed element: " + removedElement);
            notifyAll();
            return removedElement;
        }
    }
}

public class Subscriber implements Runnable {

    private final MessageQueue queue;
    private final int time;

    public Subscriber(MessageQueue queue, int time) {
        this.queue = queue;
        this.time = time;
    }

    @Override
    public void run() {
        while(true){
            try {
                queue.dequeue();
                sleep(time);
            } catch (InterruptedException e) {
                throw new RuntimeException();
            }
        }
    }
}
```

- b) (2 Punti) Si implementi una classe **Publisher** che implementi l'interfaccia **Runnable** e il cui metodo **run** inserisca in una coda **MessageQueue** un messaggio ogni **time** secondi dove **time** è una variabile di istanza inizializzata nel costruttore di **Publisher** attraverso un parametro.
- c) (2 Punti) Si implementi una classe **Subscriber** che implementi l'interfaccia **Runnable** il cui metodo **run** rimuova da una coda **MessageQueue** un messaggio ogni **time** secondi dove **time** è una variabile di istanza inizializzata nel costruttore di **Subscriber** attraverso un parametro.
- d) (1 Punto) Si scriva un **main** che crei: i) un **Publisher** x con **time = 3**, b) un **Publisher** y con **time=5**, un **Subscriber** z con **time = 2**. Si creino e lancino i relativi tre thread. Le tre istanze x, y e z devono operare su una stessa istanza di **MessageQueue**.

## Esercizio 2 – Testing (6 punti)

Si consideri il seguente frammento di codice, supponendo che tutte le variabili utilizzate siano di tipo intero:

```

1   z = 0
2   if (w > 0)
3   | z = -1
4   if (y/z == w && y > 0)
5   | if (y > 1)
6   | | z = 2
7   | else
8   | | z = 3
9 else
② ① 10 | z = 4
11 return z

```

COP. ISTRUZIONI: si, perché dopo aver coperto la linea 3 con il primo test, si potrebbe saltare questa istruzione nel secondo test, ma saltando la linea 3 si genera SEMPRE errore, in generale NO

COP. CONDIZIONI: si, perché devo eseguire almeno più di un test e, dopo aver voluto w>0 come TRUE (nessun errore), nel secondo test volteremo w>0 come FALSE e questo porterà sempre ad errore

COP. CAMMINI: si, perché ci sarà sempre almeno un cammino che non entra nell'if(w>0)

COP. DECISIONI: si, come per le condizioni si vuole perfor~~za~~ if(w>0) come FALSE

- a) (4 punti) L'istruzione alla linea 4 può generare un errore. Descrivere quali sono i criteri di testing che garantirebbero e che non garantirebbero la rilevazione di questo errore, motivando la risposta.

- b) (2 punti) Scrivere un insieme minimo di test generato dalla copertura delle istruzioni

É sufficiente 1 solo test perché le righe 5-8 non vengono mai eseguite. Vengono comunque riportati due esempi:  

w	y	z	$\frac{y}{z} = w$	$y > 0$	Return
1	-1	-1	✓	x	4
1	1	-1	x	✓	4

 Nota: bisogna comunque mettere dentro if(w>0) perché altrimenti si genererebbe errore divisione per 0

## Esercizio 3 - UML (11 punti)

Un'azienda di spedizioni vuole sviluppare un nuovo sistema informatico per tenere traccia delle richieste di spedizione degli utenti, consentire il tracciamento delle spedizioni e inviare messaggi agli utenti quando la spedizione arriva a destinazione. Le spedizioni possono essere dei seguenti cinque tipi: pacchi di dimensioni, rispettivamente, grandi, medie e piccole; lettere assicurate e normali. Ogni spedizione è associata ai dati del mittente e del destinatario.


Oltre che dai clienti dell'azienda, il sistema è usato anche dagli impiegati che devono aggiornare le informazioni di stato associate a ogni spedizione. Una spedizione può essere in uno dei seguenti stati: richiesta, presa in carico, in transito, arrivata a destinazione e bloccata.

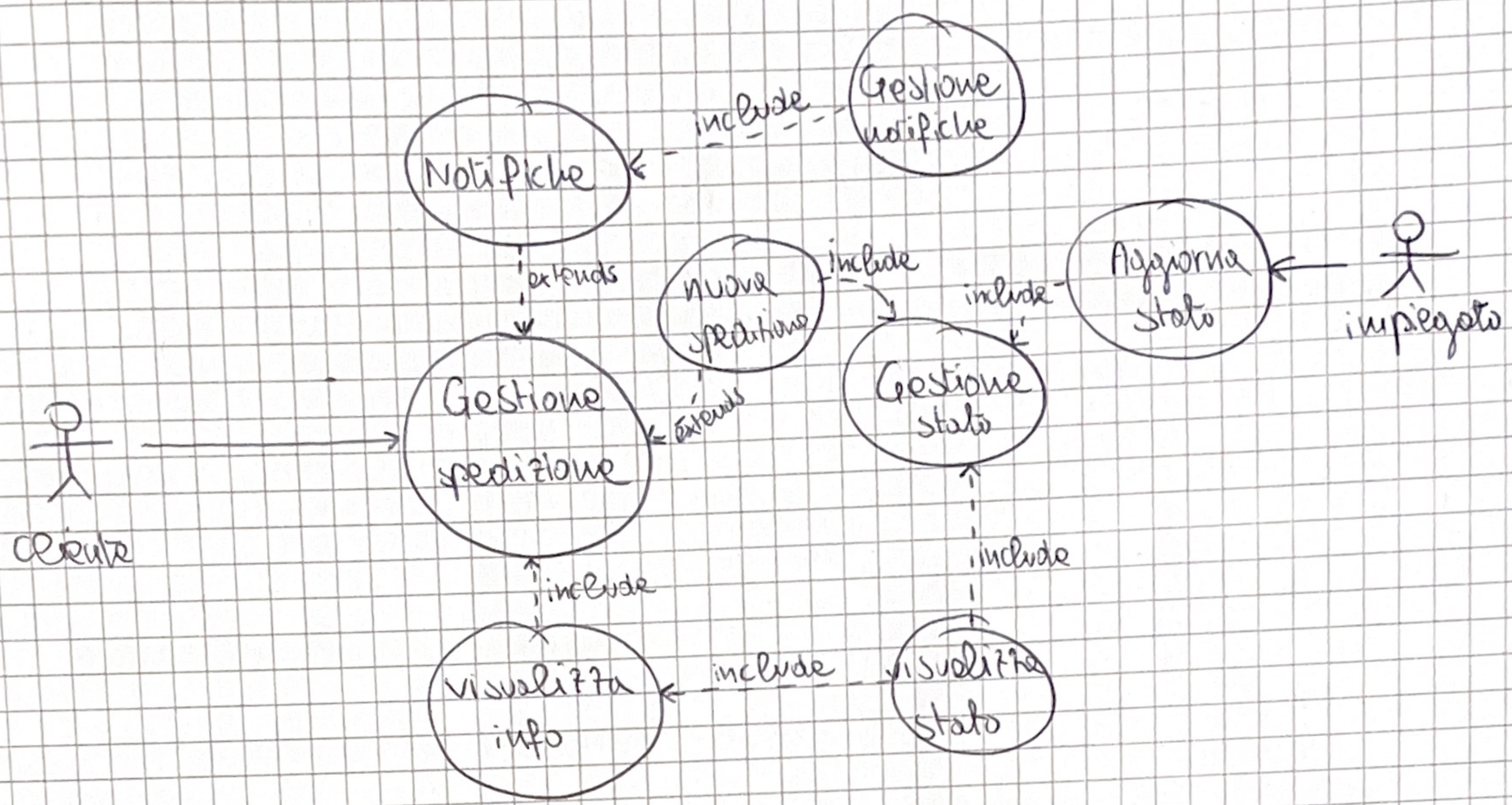
COP. ISTRUZIONI: tutte le righe di codice

DECISIONI: A if si prova reso if e reso else

CONDITIONI: A condizione ( $A \& B$ ) si prova VV VF FV FF

CAMMINI: tutti i possibili output e tutte le possibili direzioni





<<include>> : obbligatorio

`<extends>`: options

Avete ricevuto l'incarico di progettare il sistema. Vi viene richiesto di sviluppare i seguenti diagrammi UML:

- a) (2 punti) Uno use case diagram che identifichi le funzionalità principali del sistema e gli attori coinvolti.
- b) (3 punti) Una breve descrizione di ciascuno use case.
- c) (6 punti) Un class diagram che descriva le entità rilevanti del dominio applicativo e le associazioni tra queste. Si assegni un nome a ciascuna associazione e si forniscano indicazioni sui vincoli di cardinalità.

#### Esercizio 4 – Generics (5 punti)

Si consideri la classe **MessageQueue** dell'Esercizio 1.

- a) (4 punti) Si modifichi e implementi una versione della classe senza la gestione della concorrenza ma che supporti i generici in modo tale che la coda possa contenere oggetti di qualunque tipo e non solamente **String**.
- b) (1 punto) Si crei in un **main** un'istanza di **MessageQueue** che contenga numeri interi. Si implementi anche una chiamata a **enqueue** ed una a **dequeue** (l'input delle chiamate, se necessario, è a scelta).

```
public class Publisher implements Runnable{

    private final MessageQueue queue;
    private final int time;

    public Publisher(MessageQueue queue, int time) {
        this.queue = queue;
        this.time = time;
    }

    @Override
    public void run() {
        while(true){
            Random random = new Random();
            int number = random.nextInt(100);
            String text = Integer.toString(number);
            try {
                queue.enqueue(text);
                sleep(time);
            } catch (InterruptedException e) {
                throw new RuntimeException();
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        MessageQueue queue = new MessageQueue(10);

        Publisher x = new Publisher(queue, 3);
        Publisher y = new Publisher(queue, 5);
        Subscriber z = new Subscriber(queue, 2);

        Thread t1 = new Thread(x);
        Thread t2 = new Thread(y);
        Thread t3 = new Thread(z);

        t1.start();
        t2.start();
        t3.start();
    }
}
```

```
public class MessageQueue <T>{
    private List<T> data;
    private int size;

    public MessageQueue(int size) {
        data = new ArrayList<>();
        this.size = size;
    }

    public void enqueue(T message) {
        if(data.size() <= size){
            // CODA NON PIENA
            data.add(message);
            System.out.println("Added to queue: " + message);
        }
    }

    public T dequeue() {
        if(!data.isEmpty()){
            // CODA NON VUOTA
            T removedElement = data.remove(0);
            System.out.println("Removed element: " + removedElement);
            return removedElement;
        }
        return null;
    }
}

public class Main {
    public static void main(String[] args){
        MessageQueue<Integer> queue = new MessageQueue<>(12);
        queue.enqueue(10);
        queue.dequeue();
    }
}
```