



POLITECNICO
MILANO 1863

Esame di Ingegneria del Software

Corso 085885
Prof. Giovanni Ennio Quattrocchi

Data: 30-01-2023

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 (10 punti)

Si vuole implementare un sistema bancario nel quale ogni utente ha un Conto. La classe Conto gestisce una variabile float *saldo* che contiene la disponibilità sul conto. La classe Conto espone tre metodi per *prelevare*, *depositare*, e *ottenere il saldo*. L'implementazione di Conto deve tenere conto della concorrenza ed evitare che più Thread accedano in maniera incontrollata ad una istanza di Conto. La classe Banca contiene una variabile float *commissioni* ed espone un metodo *presta(Conto a, Conto b, float prestito)* attraverso il quale il proprietario del Conto *a* presta un somma uguale a *prestito* al proprietario del Conto *b*. Per fare questo:

1. Viene controllato che il saldo di *a* sia maggiore del doppio di *prestito*
2. Viene controllato che il saldo di *b* sia maggiore del triplo di *prestito*
3. In caso almeno uno dei due controlli fallisca si lancia una eccezione
4. Viene prelevata dal Conto *a* una somma uguale a *prestito*
5. Viene aggiunto il 2% di *prestito* a *commissioni*
6. Viene depositato il 98% di *prestito* sul Conto *b*

Si noti che l'insieme delle 6 operazioni deve essere eseguito in maniera transazionale ovvero non possono essere eseguite operazioni sull'istanza di Banca, su *a* e su *b* durante l'esecuzione di *presta*.

Si implementi tale sistema creando, se necessario, un insieme di classi/interfacce a supporto di questo.

Esercizio 2 – Testing (6 punti)

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

```
1  L = [-2, 4, 3, 0, 4, 5]
2  a = 0
3  if (w >= 0) {
4      if (x < 3)          ①
5          a--  a=-1
6      else if (x < 4)    ②  ③
7          a = a++ a=1
8      else                ④
9          a = 2  a=2
10 }
11 i = 0
12 while (i < a) {
```

MINIMO 3 CASI:

	w	x	y	
①	0	2	new	$L[L[-1]] \rightarrow \text{out of index}$
②	0	3	4	$L[L[1]] = 4$
③	0	10	10	Loop infinito
④	0	3	2	Loop infinito
13	if ($y > 3$) {			
14	y = 3	① ②		
15	}			
16	else			Errore un individuo: $w < 0 : L[L[0]] \rightarrow \text{out of index}$
17	++a	③ ④		ma richiederebbe un test aggiuntivo
18	i++			
19	}			
20	return $L[L[a]]$			tocca tutte le linee una volta sola

- a) (4 punti) Si definisca un insieme minimo di test utilizzando il criterio di copertura delle istruzioni.
b) (2 punti) Si descriva quali errori presenta il frammento di codice e si spieghi se tali errori vengono sempre rilevati da un insieme minimo di copertura delle istruzioni.

Esercizio 3 - UML (11 punti)

Si vuole implementare un sistema di gestione degli ordini per un negozio di vendita al dettaglio. Il sistema deve consentire ai clienti di effettuare ordini e seguire lo stato degli ordini attuali. Inoltre, il sistema deve essere in grado di gestire diverse tipologie di ordini e di pagamento. Per quanto riguarda le personalizzazioni degli ordini, ad esempio, un cliente potrebbe voler aggiungere un'opzione di consegna express o di imballaggio regalo. Per quanto riguarda gli stati degli ordini, ad esempio, un ordine può passare attraverso diversi stati come "In attesa di pagamento", "In preparazione", "In consegna" e "Consegnato". Il sistema deve essere in grado di gestire questi cambiamenti di stato in modo flessibile e facilmente estendibile per eventuali future modifiche.

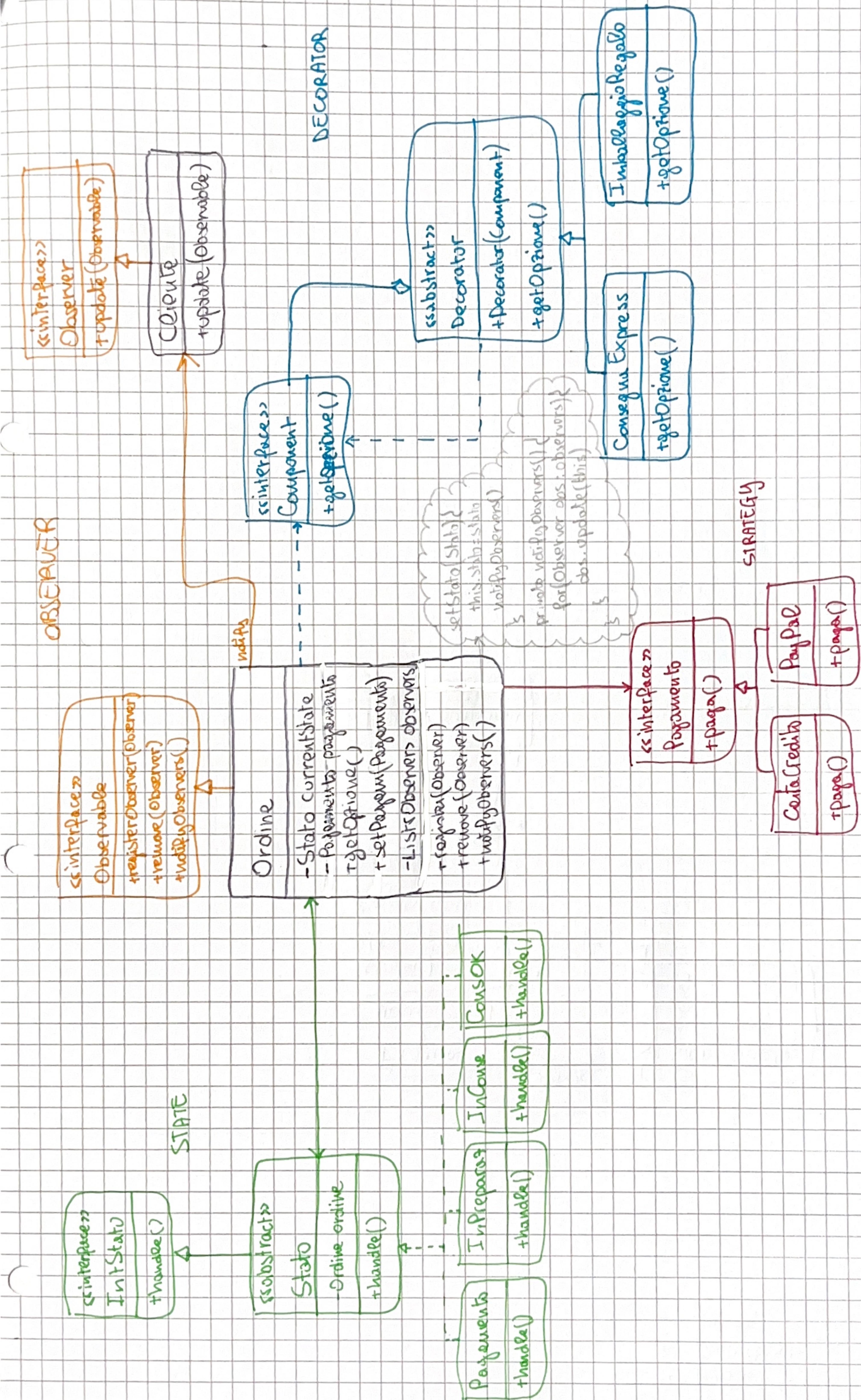
- a) (6 punti) Si definisca un Class Diagram, utilizzando opportuni design pattern, che descriva le entità rilevanti del dominio applicativo, indicando metodi e attributi significativi e le associazioni tra entità. Si assegni un nome a ciascuna associazione e si forniscano indicazioni sui vincoli di cardinalità. Si commentino le scelte effettuate quando necessario.
b) (5 punti) Per uno dei design pattern utilizzati si riporti il relativo Class Diagram astratto e si spieghi brevemente quali problemi risolve

Esercizio 4 – Generics (6 punti)

Si vuole definire la classe Dieta con un singolo parametro di tipo generico T (che rappresenta il tipo di alimento). La classe Dieta contiene una variabile membro privata di tipo Map<String, List<T>> per memorizzare il programma settimanale, dove le chiavi rappresentano i giorni della settimana (stringhe) e i valori rappresentano la lista di alimenti per quel giorno.

Si implementi:

- Un costruttore che inizializza la dieta con una mappa vuota.
- Un metodo aggiungiAlimento che aggiunge un alimento alla dieta per un giorno specifico.
- Un metodo chiamato rimuoviCibo che rimuove un alimento dalla dieta per un giorno specifico
- Un metodo chiamato ottieniPasto che restituisce la lista di alimenti per un giorno specifico
- Un metodo ottieniDieta che restituisce l'intera dieta come una mappa
- Infine, creare un metodo main che crea due istanze della classe Dieta che utilizzano rispettivamente String e Alimento per rappresentare gli alimenti. Alimento è una semplice classe Java che contiene il nome dell'alimento e le relative calorie (int). Per ciascuna istanza di Dieta si aggiungano due alimenti a piacere.



```
public class Account {
    private float amount;

    public Account(float amount) {
        this.amount = amount;
    }

    public void deposit(float amount) {
        this.amount += amount;
    }

    public void withdraw(float amount) {
        this.amount -= amount;
    }

    public float getAmount() {
        return amount;
    }
}

public class Bank {
    private float totalCommissions;

    public Bank() {
        this.totalCommissions = 0;
    }

    // A lends to B
    public void lend(Account a, Account b, float loan) {
        // Lock objects
        Account firstLock = a;
        Account secondLock = b;

        // Order accounts on hashCode to avoid deadlock
        if (System.identityHashCode(a) > System.identityHashCode(b)) {
            firstLock = b;
            secondLock = a;
        }

        synchronized(firstLock) {
            synchronized(secondLock) {
                if (a.getAmount() >= 2 * loan && b.getAmount() >= 3 * loan) {
                    a.withdraw(loan);
                    float loanCommissions = loan * 0.2f;
                    this.totalCommissions += loanCommissions;
                    float movingMoney = loan - loanCommissions;
                    b.deposit(movingMoney);
                } else {
                    throw new IllegalArgumentException();
                }
            }
        }
    }
}
```

```
public class Diet <T>{
    // T is the food type

    Map<String, List<T>> weekProgram;

    public Diet(){
        weekProgram = new HashMap<>();
    }

    public void addFood(T food, String day){
        List<T> foodList = weekProgram.get(day);
        if(foodList == null){
            foodList = new ArrayList<>();
            weekProgram.put(day, foodList);
        }
        foodList.add(food);
    }

    public void removeFood(T food, String day){
        List<T> foodList = weekProgram.get(day);
        if(foodList != null){
            foodList.remove(food);
        }
    }

    public List<T> getDailyFood(String day){
        List<T> foodList = weekProgram.get(day);
        if (foodList == null) {
            return new ArrayList<>();
        }
        return foodList;
    }

    public Map<String, List<T>> getWeekProgram(){
        return weekProgram;
    }

    public static void main(String[] args) {
        Diet<String> dietOne = new Diet<>();
        Diet<Food> dietTwo = new Diet<>();

        dietOne.addFood("Fish", "Monday");
        dietOne.addFood("Meat", "Tuesday");

        dietTwo.addFood(new Food("Cookies", 5000), "Wednesday");
        dietTwo.addFood(new Food("Cake", 10000), "Thursday");
    }
}
```