Corso di laurea in Informatica, Dipartimento di Informatica - Università di Pisa

Corso di Ingegneria del Software, Laura Semini, **Quarto Appello 2022-23, 7 giugno 2023**

Fino a qui tutto bene

Domanda 1 Descrivere con un diagramma UML tutti i casi d'uso del Sistema. Per uno di essi dare la narrativa usando un diagramma di sequenza

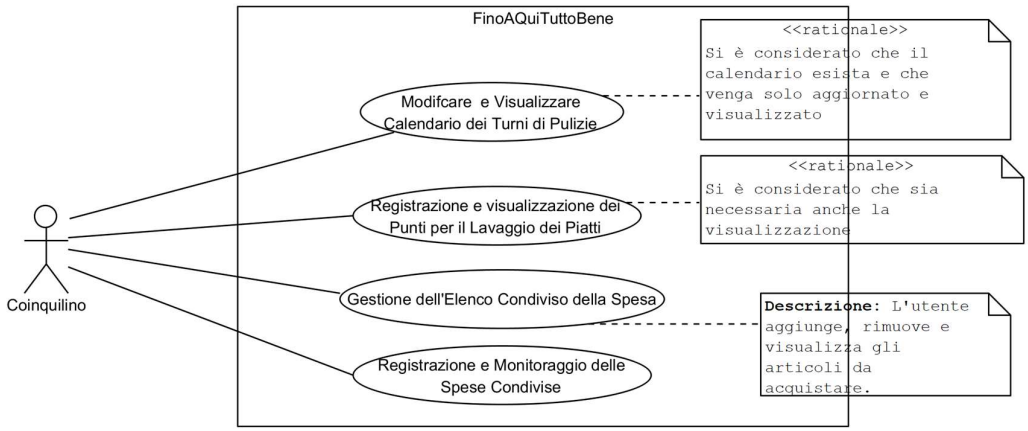
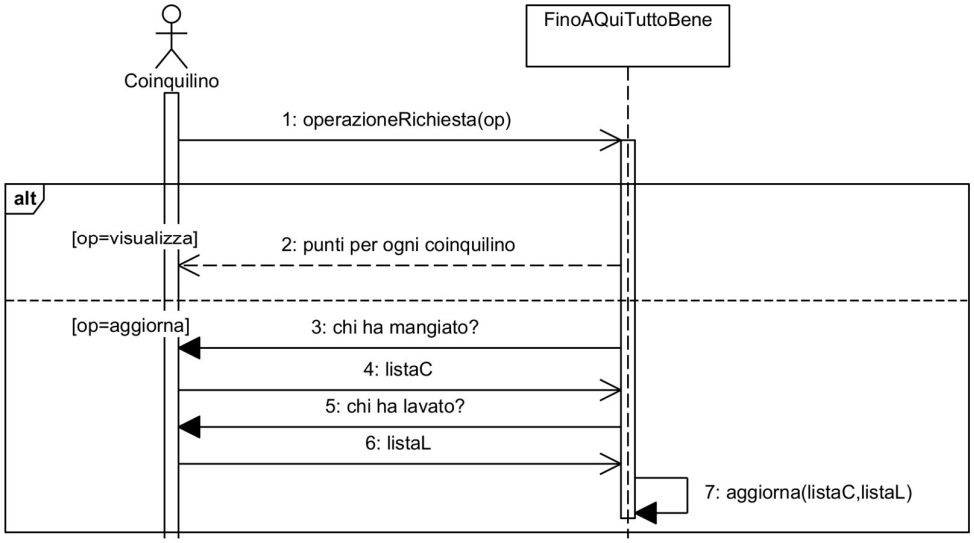
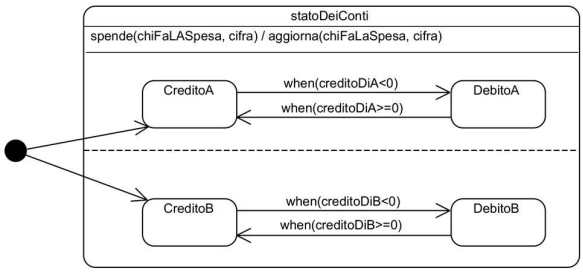


Diagram name Registrazione e visualizzazione dei Punti per il Lavaggio dei Piatti



Domanda 2 Usando un diagramma di macchina a stati, descrivere l'evoluzione di un oggetto che mantiene lo stato dei conti in un appartamento di 2 studenti. Lo stato del conto è rappresentato come uno stato composito parallelo, in cui ogni studente ha due stati distinti: "A_Credito" e "A_Debito". Per semplicità lo stato di parità viene modellato come se fosse a credito. Non sono previsti pagamenti

studente-studente, ma solo pagamenti per pagare la spesa o le utenze.



Domanda 3. Usare il pattern Singleton per gestire la creazione e l'accesso a un oggetto "CalendarioTurni".

```
//soluzione che usa eager initialization

public class CalendarioTurni {

    // Istanza unica del CalendarioTurni
    private static final CalendarioTurni instance = new CalendarioTurni();

    // Altri attributi del CalendarioTurni
    private String[] turni;

    // Costruttore privato
    private CalendarioTurni() {
        // Inizializzazione degli attributi del CalendarioTurni
    }

    // Metodo per ottenere l'istanza unica
    public static CalendarioTurni getInstance() {return instance;}

    // Altri metodi del CalendarioTurni: visualizzazione e modifica
    public String[] getTurni() {return turni;}
    public void modificaCalendario(String[]nuoviTurni){turni = nuoviTurni;}
}
```

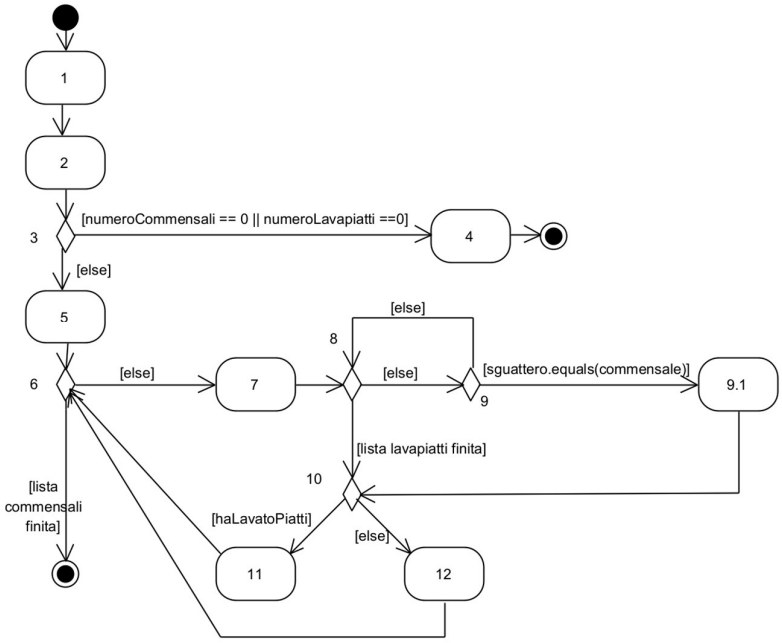
Il metodo `finePasto()` aggiorna lo stato dei punti come segue: ogni commensale che ha mangiato senza lavare i piatti perde un punto, chi ha lavato i piatti guadagna un numero di punti pari a (numero commensali - numero lavapiatti)/numero lavapiatti

```
public void finePasto(String[] commensali, String[] lavapiatti, StatoPunti
statoPunti) {
   1. int numeroLavapiatti = lavapiatti.length;
   2. int numeroCommensali = commensali.length;
   3. if (numeroCommensali == 0 || numeroLavapiatti ==0)
   4.       throw new IllegalArgumentException("array vuoto");
   5. int puntiGuadagnati = (numeroCommensali - numeroLavapiatti) /
      numeroLavapiatti;

   6. for (String commensale : commensali) {
   7.       boolean haLavatoPiatti = false;
   8.       for (String sguattero: lavapiatti)
   9.          if (sguattero.equals(commensale)) {
                      1. haLavatoPiatti = true;
                      2. break;
              }
   10.      if (haLavatoPiatti)
```

```
11.          statoPunti.aggiorna(commensale, puntiGuadagnati);
         else
12.          statoPunti.aggiorna(commensale, -1);
     }
}
```

Domanda 4. Disegnare il diagramma di flusso di `finePasto()` usando un opportuno diagramma UML.



Domanda 5. Fornire una proof obligation che si basa su criteri white box.

Copertura di comandi e decisioni:
Invocare il metodo `finePasto` con:

1. una lista di commensali vuota o una lista di lavapiatti vuota (caso di errore)
2. una lista di commensali non vuota e contenente almeno un lavapiatti e uno che non li ha lavati e una lista di lavapiatti non vuota

Bastava un critetio di copertura, il resto della soluzione è opzionale o alternativa

Per avere copertura delle condizioni semplici (copre anche le condizioni composte)

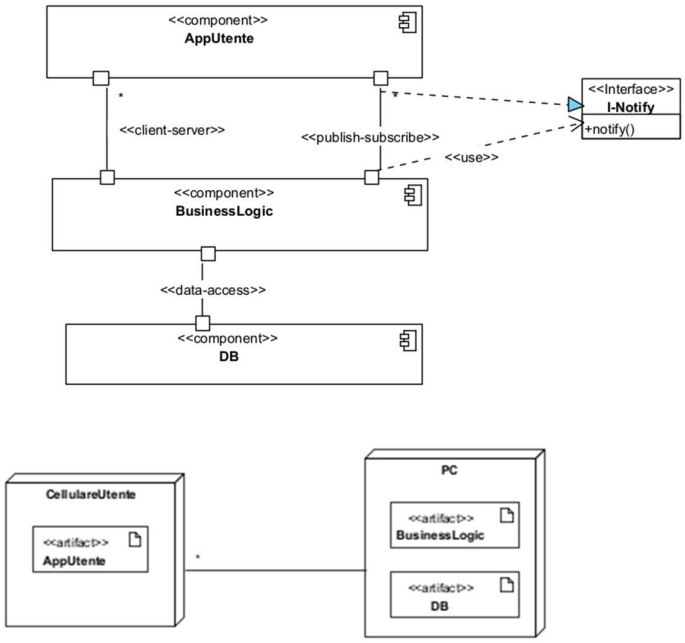
1. una lista di commensali vuota
2. una lista di commensali vuota e una lista di lavapiatti vuota
3. una lista di commensali non vuota e contenente almeno un lavapiatti e uno che non li ha lavati e una lista di lavapiatti non vuota

Per avere copertura dei cammini? (in aula)

Esercizi Extra

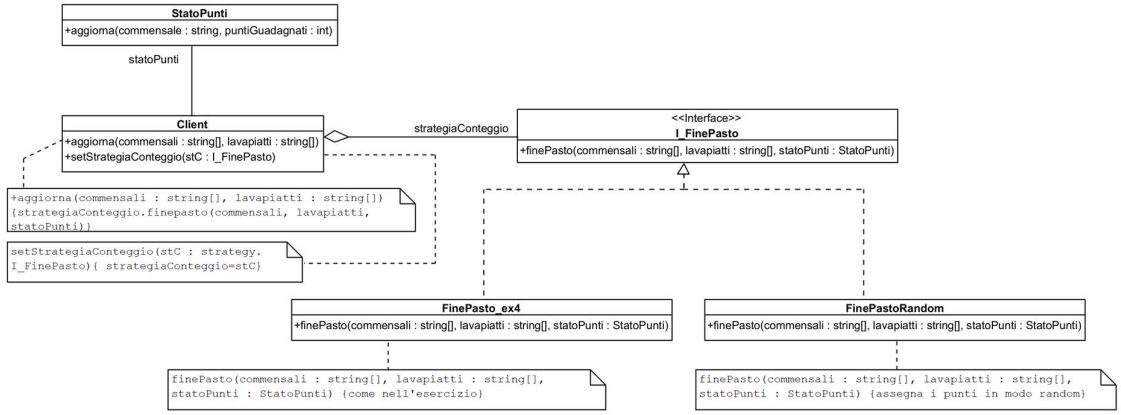
Domanda 6. Dare una vista C&C e una di dislocazione. Si faccia attenzione al requisito: L'applicazione deve fornire notifiche agli utenti quando sono a debito o quando gli altri utenti hanno effettuato pagamenti che li riguardano.

Soluzione costruita in aula in cui, rispetto alla consegna, abbiamo anche aggiunto un'interfaccia:



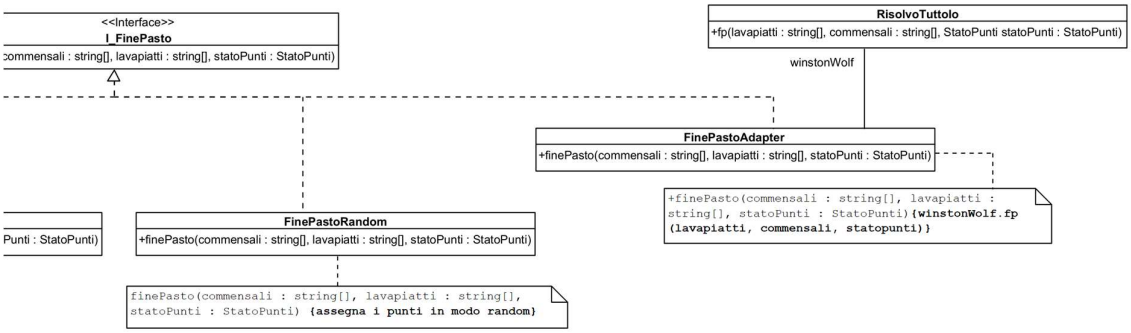
Domanda 7. Siccome l'implementazione del metodo `finePasto(String[] commensali, String[] lavapiatti, StatoPunti statoPunti)` fornita sopra ha fatto scoppiare liti furibonde in appartamento, si è deciso di fornire diverse implementazioni e lasciare ai condomini decidere quale usare. Che design pattern si usa? Immaginare un'altra possibile implementazione e dare lo schema delle classi coinvolte.

Strategy:



Domanda 8. Il problema dell'implementazione del metodo `finePasto(String[] commensali, String[] lavapiatti, StatoPunti statoPunti)` continua ad essere un problema. Uno dei coinquilini ha scoperto la classe `RisolvoTuttoIo` che implementa il medodo `fp(String[] lavapiatti, String[] commensali, StatoPunti statoPunti.` Per rispettare l'open closed principle, non si modifica il codice progettato nella precedente domanda, ma si vuole poter usare anche il metodo `fp` della classe `RisolvoTuttoIo`. Che design pattern si usa? Dare lo schema delle classi coinvolte.

Adapter:



Domanda 9. Il metodo `finePasto(String[] commensali, String[] lavapiatti, StatoPunti)` aggiorna lo stato dei punti come segue: ogni commensale che ha mangiato senza lavare i piatti perde un punto, chi ha lavato i piatti guadagna un numero di punti pari a (numero commensali - numero lavapiatti)/numero lavapiatti. Quali strategie di progettazione di casi di test a scatola chiusa adotteresti?

Domanda 10. Dato il metodo `finePasto(String[] commensali, String[] lavapiatti, StatoPunti)` quali strategie di testing combinatorio adotteresti?