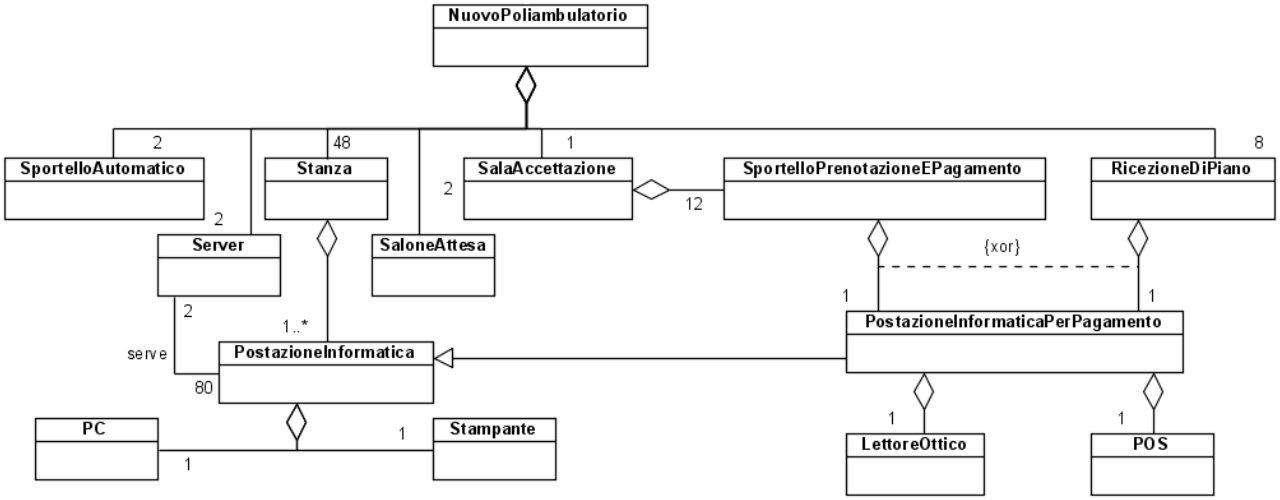
Corso di Ingegneria del software – Primo Appello – 14 gennaio 2010

C, Montangero, L. Semini
Dipartimento di Informatica, Università di Pisa, a.a. 2009/10

La prova si svolge a libri chiusi (non è permessa la consultazione di materiale didattico).
Traccia delle soluzioni

Si consideri il caso di studio Poliambulatorio.

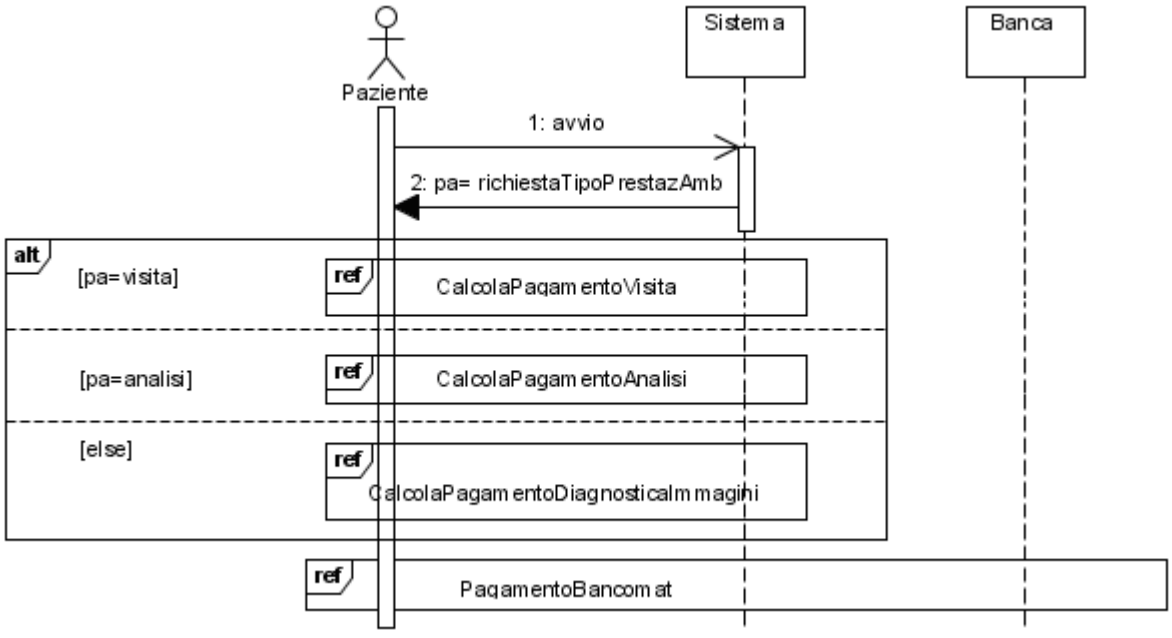
Domanda 1. Analisi del dominio.
Descrivere la struttura e la dotazione hardware del nuovo Poliambulatorio dell'Ospedale Maggiore di Milano, usando un diagramma delle classi.
Risposta. Un possibile diagramma è il seguente.



Si consideri che POS e lettore ottico hanno bisogno di una macchina cui appoggiarsi.

****

Si consideri il caso d'uso **Gestione sportelli automatici POS per il pagamento del ticket**. Viene fornito il seguente diagramma di sequenza che mostra come il sistema realizza il caso d'uso.



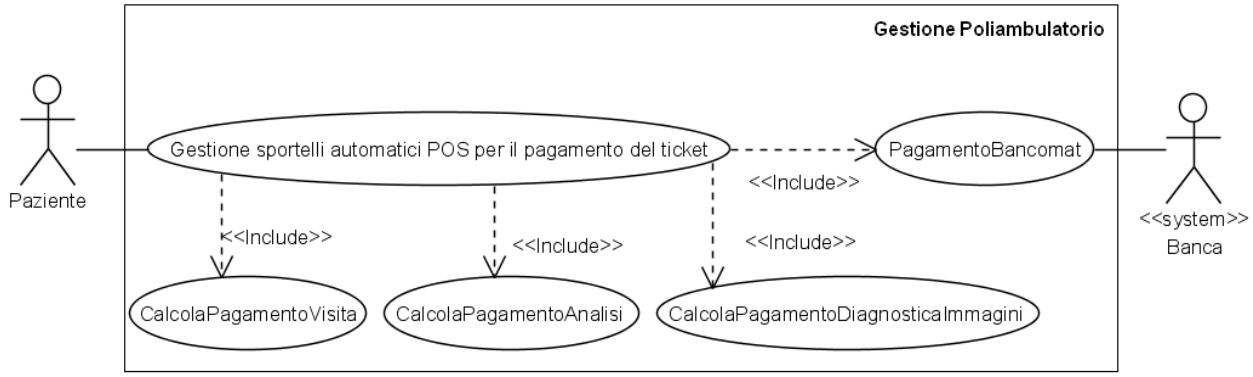
Domanda 2. Requisiti.

a) Dare un diagramma dei casi d'uso per **Gestione sportelli automatici POS per il pagamento del ticket**.

b) Dare la narrativa di **Gestione sportelli automatici POS per il pagamento del ticket**.

Risposta.

a)



b)

Nome: Gestione sportelli automatici POS per il pagamento del ticket.
Breve descrizione: Il sistema calcola la quota a carico dell'assistito e permette il pagamento bancomat della stessa.
Attore principale: Paziente.
Attori secondari:Nessuno.
Precondizioni: NA
Postcondizioni: Ticket pagato.
Sequenza principale degli eventi:

1. Il paziente avvia il caso d'uso.
2. Il sistema chiede che tipo di prestazione ambulatoriale intende pagare;
3. **se** (visita)
   3.1. **include** CalcolaPagamentoVisita
4. **altrimenti se (**analisi**)**
   4.1. **include** CalcolaPagamentoAnalisi
5. **altrimenti**
   5.1. **include** CalcolaPagamentoDiagnosticaImmagini
6. **include** PagamentoBancomat

Situazioni eccezionali: Fallimento di uno dei casi d'uso inclusi.

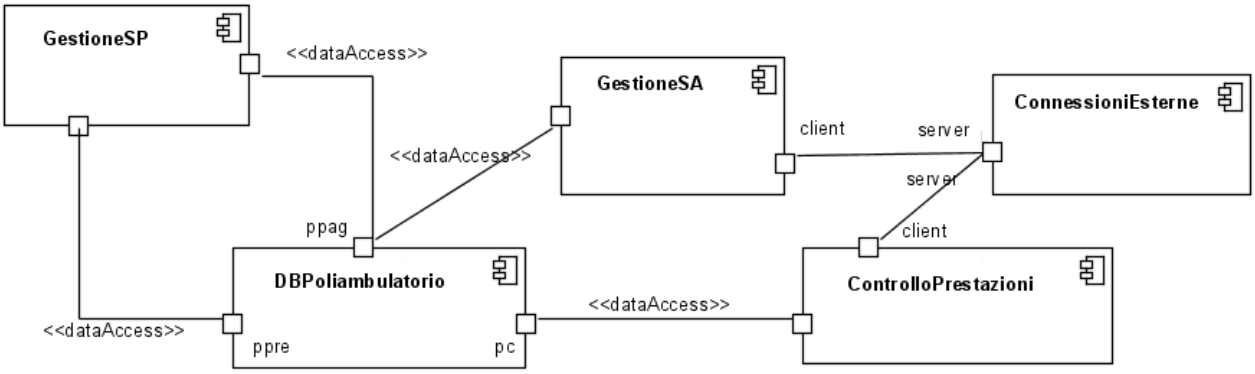
****

Si consideri il sottosistema amministrativo, in particolare i punti 2 e 3, che danno luogo ai casi d'uso "Gestione sportelli automatici POS per il pagamento del ticket" e "Controllo pagamento delle prestazioni erogate". Si considerino inoltre le seguenti componenti:

Componente	**Responsabilità**
GestioneSA	Gestisce gli Sportelli Automatici POS per il pagamento del ticket.
GestioneSP	Permette prenotazioni e pagamento del ticket agli sportelli presidiati.
DBPoliabulatorio	Mantiene i dati del poliambulatorio, in particolare prenotazioni e relativi pagamenti.
ControlloPrestazioni	Controlla il pagamento delle prestazioni erogate, o comunque prenotate e non disdette entro 48 ore, e individua la quota a carico di altre regioni, se del caso.
ConnessioniEsterne	Gestisce le connessioni con l'anagrafe di Milano, la regione Lombardia e con l'ASL di residenza del paziente.

Domanda 3. Architettura.
Fornire una vista C&C dell'architettura del sotto-sistema considerato, sapendo che DBPoliambulatorio ha tre diversi porti: ppag, per registrare i pagamenti;
ppre, per registrare le prenotazioni;
pc, per query di controllo pagamenti.

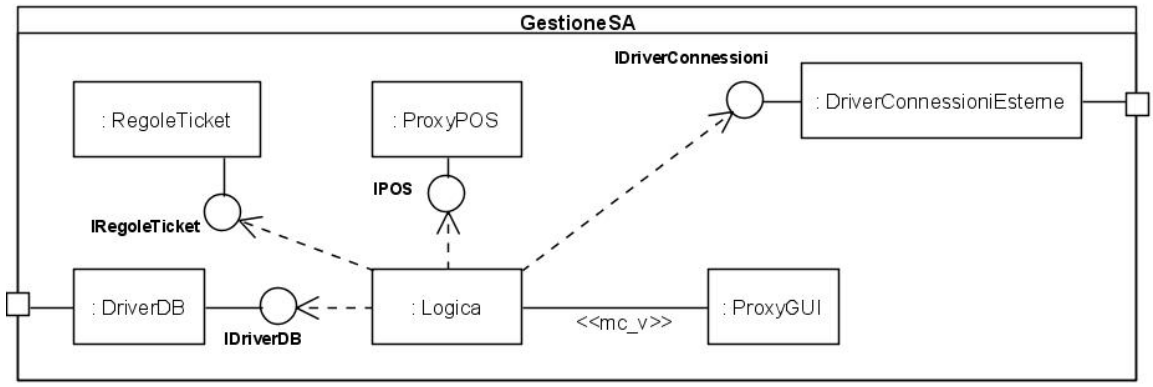
Risposta.



****

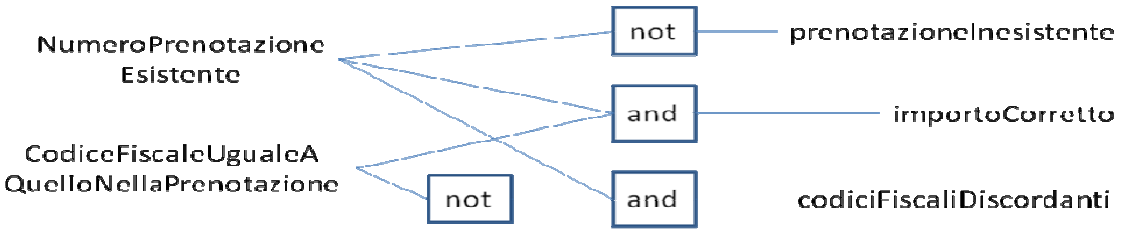
Domanda 4. Progettazione di dettaglio. Dare un diagramma di struttura composita per la componente GestioneSA. Oltre alla Logica, per facilitare la manutenzione adattativa del sistema, è stata individuata una parte, RegoleTicket, che mantiene le regole per il calcolo del ticket. Gestione SA comunica con altre componenti, come mostrato nella vista C&C, e con l'interfaccia utente e il POS.

Risposta.



****

Si consideri il seguente grafo di causa/effetto, costruito per descrivere quando l'importo del ticket di una visita ambulatoriale è determinato correttamente, in funzione dei dati forniti dall'assistito – numero prenotazione e codice fiscale (N.B.: stiamo trascurando l'accesso all'anagrafe degli assistiti):



Si considerino anche le seguenti classi, che mostrano parte della realizzazione del calcolo del ticket di una visita ambulatoriale, nel contesto del progetto dell'esercizio precedente:

```
public interface IDriverDB {
   Prenotazione getPrenot(Integer numeroPrenot)
                              throws PrenotazioneInesistenteEx;
    //...
}
public class Ticket {
    protected Euro importo = Euro.zero;
    //...
}
public class TicketVisitaAmbulatoriale extends Ticket {

// parte di configurazione
    private static IDriverDB driverDB = null;
    private static IRegoleTicket regole = null;
    public static void inizializza(IDriverDB dDB, IRegoleTicket r){
        driverDB = dDB;
        regole = r;
    }
// parte operativa
    public TicketVisitaAmbulatoriale(Integer numPrenot,
                                     CodiceFiscale cf  )    {
        Prenotazione prenot = driverDB.getPrenot(numPrenot); //1
        boolean concordanzaCF = prenot.haComeCF(cf);         //2
        if (!concordanzaCF) {                                        //3
            throw new CFdiscordantiEx(prenot, cf);           //4
        }
        importo = regole.determinaImporto(prenot);           //5
    }
   // ...
}
```

Domanda 5. (Verifica)

a) Una batteria di test che realizzi il grafo dato sopra, quanto copre del codice di `TicketVisita-Ambulatoriale`, secondo il *criterio delle decisioni*? Giustificare la risposta, usando i numeri nei commenti. **N.B.** Si tenga conto che la possibilità che la chiamata di `getPrenot` sollevi un'eccezione è una decisione implicita nel codice.

b) Cosa deve garantire l'ambiente di test, rispetto alla relazione temporale tra le invocazioni di `inizializza` e `TicketVisitaAmbulatoriale`**?**

Risposta:

a) Abbiamo due decisioni, una alla riga 1 (implicita: getPrenot ritorna regolarmente) e una alla riga 3 (i codici fiscali non concordano). La copertura è del 100%, come mostrato dalla tabella a lato.

b) Ovviamente, `inizializza` deve essere invocato prima di `TicketVi-sitaAmbulatoriale`.

		getPrenot ritorna regolarmente	
		vero	Falso
i codici fiscali non concordano	non valutata		prenotazione Inesistente
	Vero	codici Fiscali Discordanti	
	Falso	importo Corretto	