Corso di Ingegneria del Software – Appello del 1^ giugno 2016

V. Gervasi, L. Semini - Dipartimento di Informatica, Università di Pisa, a.a. 2015/16

Nome_____ Cognome_____

Matricola_____ Corso _____

Aula_____ Posizione nell'aula_____

(senza contare i banchi vuoti)

			cattedra			
A1	A2	A3	A4	…	…	…
B1						
C1						
….						

Si consideri il seguente caso d'uso: PRENOTAZIONE TRAMITE APP
Attore primario: Cliente
Attori secondari: Autista, Istituto di credito
Precondizioni: Cliente autenticato tramite l'app
Postcondizioni: Prenotazione effettuata oppure richiesta al Cliente di riprovare più tardi.
Sequenza principale degli eventi:

1. Il Cliente chiede, tramite l'app, l'invio di un'auto, specificando tipo, indirizzo, orario e carta di credito con cui pagare la corsa.
2. Il Sistema trasmette la richiesta a tutti gli autisti in servizio, dotati di un'auto del tipo richiesto, e che non siano già assegnati ad altre corse nell'orario indicato
3. Il sistema raccoglie le segnalazioni di disponibilità degli autisti nell'arco di 1 minuto
4. **Se** (l'insieme degli autisti disponibili non è vuoto), il sistema sceglie uno degli autisti disponibili (in base a diverse euristiche) e gli assegna la corsa, informandolo.
5. **Altrimenti**, se possibile, si ripete dal punto 2. con il tipo di auto successivo, in ordine di prezzo crescente (ma specificando nella richiesta che per la corsa si offre il prezzo corrispondente al tipo di auto della richiesta originale).
6. **Se** (è stato trovato un autista)
   6.1. Il Sistema informa il Cliente sui tempi di attesa
   6.2. Il Cliente accetta la corsa
   6.3. Il Sistema assegna la corsa all'autista e lo informa
   6.4. Il Sistema richiede all'Istituto di credito la pre-autorizzazione ad addebitare sulla carta di credito l'importo della corsa.
   6.5. Il sistema conferma la corsa all'utente, fornendo i dettagli di contatto dell'autista a cui è stata assegnata.
7. **Altrimenti** il sistema chiede al cliente di riprovare dopo qualche minuto

Sequenze alternative degli eventi: Non ci sono autisti disponibili. L'istituto di credito rifiuta la pre-autorizzazione.

Domanda 1. Dare un diagramma delle classi coinvolte nel caso d'uso.

Domanda 2. Valutare se è più adeguato un diagramma di macchina a stati o un diagramma di attività per modellare la fase di prenotazione, e fornire il diagramma scelto.

Domanda 3. Si diano i diagrammi di contesto secondo il metodo Jackson con i domini rilevanti per il caso d'uso, fornendo diagrammi separati per il caso in cui si sia chiamati a sviluppare soltanto:

1. la app in dotazione agli utenti
2. il Sistema REBU
3. il sistema di interfaccia (verso REBU) dell'Istituto di credito

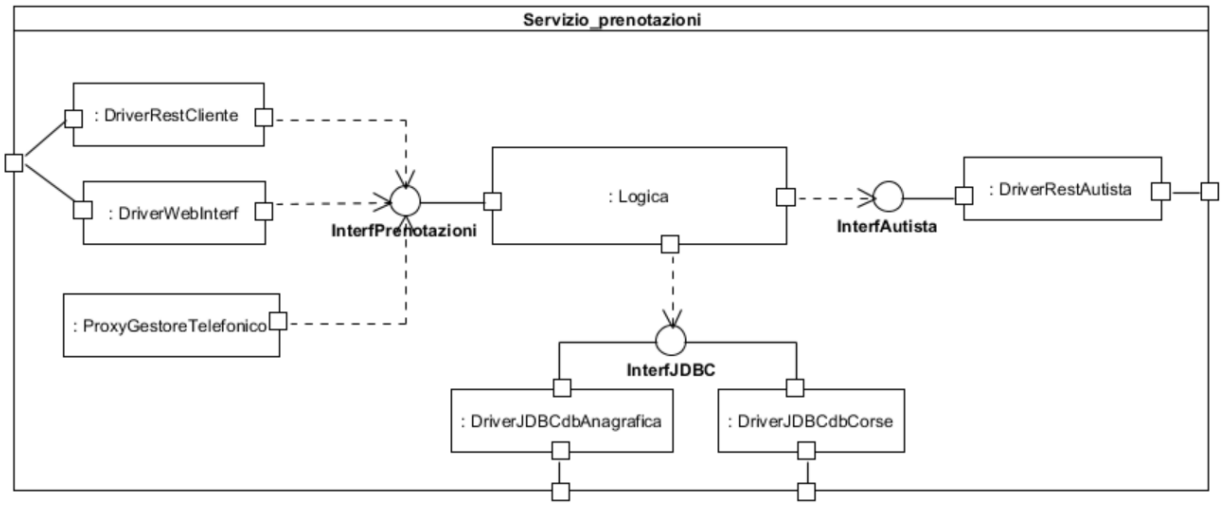
In particolare, si indichi per ciascun caso quali domini sono dati (given), quali progettati (designed), e quali costituiscono la macchina (machine).

Domanda 4. Si assuma che sia stata definita un'architettura con le seguenti componenti (l'architettura realizza anche i casi d'uso per la prenotazione via Web o SMS):

1. App_cliente (che implementa l'app usata dal cliente)
2. Sistema_prenotazione (che implementa la business logic del sistema)
3. DB_anagrafica (che raccoglie dati anagrafici su autisti e clienti)

4. DB_corse (che raccoglie dati su prenotazioni, corse assegnate, corse effettuate)
5. App_autista (che implementa l'app usata dall'autista)
6. Interfaccia_web (che realizza un'interfaccia web per i clienti)

e che la progettazione di dettaglio abbia portato a definire la seguente struttura interna per la componente sistema_prenotazione, Il progetto di dettaglio tiene conto del fatto che la componente sistema_prenotazione oltre a un'interfaccia basata su REST alle app, e voglia accettare anche le operazioni richieste via SMS o dal sito web. Le richieste via SMS sono gestite dal provider scelto (TIM, Vodafone…) che comunica tramite un proprio protocollo con l'applicazione.



Dare la vista strutturale ibrida di decomposizione e degli usi delle parti che realizzano la componente. Ogni parte deve essere nel package dell'interfaccia che realizza.

Domanda 5. Le auto di REBU, circolando per molte ore nei centri cittadini, devono superare ogni anno un rigido test sui valori delle emissioni degli ossidi di azoto (NOx). Il metodo:

```
public void calcolaGiriMotore (double valoreSensoreAcceleratore)
```

dato un valore ricevuto dall'acceleratore dell'auto, calcola il numero di giri del motore, salva il risultato in un file di log e ordina al motore di girare a quel numero di giri. La centralina delle automobili implementa il metodo e lo invoca ogni decimo di secondo leggendo da un sensore sull'acceleratore.

In officina, la strumentazione di misura delle emissioni viene collegata via cavo alla centralina. Le emissioni vanno lette per tre soglie date di numero di giri. Per ogni soglia s_i, il meccanico accelera fino a quando la strumentazione di misura dice "ok" perché legge s_i dal file di log. A questo punto la strumentazione di misura raccoglie il gas di scarico per le analisi.

Si consideri la seguente implementazione (alcune parti sono solo commentate per semplicità:

```
public void calcolaGiriMotore (double valoreSensoreAcceleratore){
      int numeroGiri, numeroGiriFake;
      \\ per una opportuna funzione f
      numeroGiri = f(valoreSensoreAcceleratore);

      if(inMovimento(ruoteMotrici)&!inMovimento(ruoteNonMotrici))
            numeroGiriFake = 0.7 * numeroGiri;
            else numeroGiriFake = numeroGiri;
      … // scrivi numeroGiri nel file di log;
      … // invia numeroGiriFake al motore;
}
```

1. Si dia una definizione di difetto latente alla luce di questo esempio;
2. Supponendo di avere a disposizione il codice sorgente e di poter applicare un criterio a scatola aperta, disegnare il grafo di flusso del metodo e dare un insieme minimo di valori restituiti dallo stub che realizza il metodo `inMovimento()` per avere copertura al 100% delle decisioni;