Linguaggi di Programmazione per la Sicurezza

Progettazione di Software Sicuro

Progettazione/Ingegneria del Software (prima parte)

Esame del 2 Febbraio 2016 – (Parte Scritta)

1. Descrivere (a) il modello a spirale ed (b) il motivo per cui il modello a spirale sia ritenuto idoneo per lo sviluppo di software sicuro. **[pt. 3]**

2. Descrivere i limiti delle FSM di base e come questi sono superati nelle sue estensioni. **[pt. 3]**

3. (a) Dare la definizione di una FSM di Mealy temporale. (b) Portare un semplice esempio. [**pt.3**]

4. Utilizzando il formalismo delle macchine di comunicazione temporali, modellare il comportamento del seguente sistema di pagamento di pedaggio autostradale tramite telepass. All'arrivo di un'auto al casello autostradale, comunicata tramite sensore, un computer invia al sistema telepass l'informazione della targa della macchina e l'importo da pagare. Il sistema, fatti i dovuti controlli, invia al computer accettazione o rifiuto della richiesta. Ricevuta la risposta, il computer invia un segnale sul display di rifiuto del pagamento, oppure il comando di apertura alla sbarra. Questa provvede ad aprirsi ed a richiudersi dopo un minuto a seguito di opportuno comando del computer. [**pt. 5**]

5. (a) Descrivere a cosa servono precondizioni, postcondizioni ed invarianti nel Design by Contract. (b) Scrivere la segnatura e l'opportuno contratto JML per un metodo che data una sequenza di interi, la ripartisca in due sequenze di interi positivi ed interi negativi. [**pt. 3**]

6. Calcolare la test suite per testare tutte le condizioni secondo l'MCDC nel seguente statement di `if` annidati:

```
if (x > 5 & y > 0) | (z > 5) them
        if (z > 5 & v < z) then …
```

[pt. 3]

Progettazione del software e Ingegneria del SW
Esame del 2 febbraio 2016 (Prova di laboratorio)

Istruzioni

- Creare una cartella sul desktop con il proprio **cognome**.
- Aprire eclipse: Start/Tutti i Programmi/Ingegneria del software/Eclipse
- Selezionare come workspace la cartella creata sul Desktop. Se eclipse si avvia senza far scegliere il workspace, una volta avviato impostare File/Switch Workspace/Other...: a questo punto dovrebbe essere possibile selezionare il workspace.

  ATTENZIONE: se il workspace non è sul desktop tutto il lavoro sarà perso e non recuperabile.
- Creare un progetto Java e denominarlo con il proprio **numero di matricola**.
- Al termine del compito, **non** chiudere eclipse e chiamare il docente per consegnare.
- Consegnare solo la cartella del progetto Java. Dentro la cartella creata sul desktop (quella con il cognome) c'è un'altra cartella (quella con il numero di matricola) che corrisponde al progetto Java: consegnare quest'ultima cartella. Chiudere eclipse solo dopo aver consegnato.

1 Fila [pt. 2]

Scrivere una classe *Fila* per modellare una fila ad uno sportello. Ogni utente ritira un numero da 0 a 199 per effettuare un'operazione allo sportello. Ogni operazione corrisponde a un numero stimato di *minuti* (da 1 a 15). Gli operatori allo sportello chiamano il prossimo numero, in ordine crescente a partire da 0 fino a 199. Il primo utente della giornata prende il numero 0. Ogni nuovo utente prende il numero successivo all'ultimo ritirato. Al massimo sono serviti 200 utenti.

La fila deve essere rappresentata da un array (`fila[]`) da 0 a 199 dove ogni cella di indice i corrisponde al numero di minuti stimati (intero da 1 a 15) per l'operazione dell'utente che ha preso il numero i. Due variabili, che vanno da 0 a 199, contengono il numero del primo posto della fila libero (`prossimoInFila`) e il prossimo utente che sarà servito (`prossimoServito`). Per esempio, se la fila è composta da cinque utenti (da 0 a 4) e l'utente 1 è allo sportello si ha che `prossimoServito` è 2 e `prossimoInFila` è 5.

Il costruttore della classe deve creare e inizializzare una fila vuota (con minuti tutti a 0) e le due variabili `prossimoServito` e `prossimoInFila` in modo coerente.

Metodo ritiroNumero La classe dispone di un metodo `ritiroNumero(int minuti)` che restituisce il numero di fila all'utente e inserisce nella fila i `minuti` stimati per l'operazione. Nel caso siano già stati inseriti in fila 200 utenti il metodo restituisce -1. Il metodo è mostrato in Codice 1.

```
public int ritiroNumero(int minuti) {
    int numero = -1;
    if(prossimoInFila<200 && prossimoInFila>=0) {
        fila[prossimoInFila] = minuti;
        numero = prossimoInFila++; }
    return numero;
}
```

Codice 1: Metodo `ritiroNumero(int minuti)`

Metodo chiamataAlloSportello La classe deve disporre di un metodo `chiamataAlloSportello()` che ritorni un intero (ovvero il prossimo numero da servire e che quindi viene chiamato allo sportello) e che aggiorni di conseguenza `prossimoServito`. Nel caso in cui la fila sia finita (ovvero non ci sono utenti in attesa o stati stati serviti tutti e 200 gli utenti previsti) il metodo deve restituire -1.

Metodo stimaMinuti La classe deve disporre di un metodo `stimaMinuti(int numero)` che, dato un numero in fila, ritorni un intero che contiene la stima del numero di minuti che deve aspettare l'utente prima che tocchi a lui (senza considerata l'utente allo sportello). Per esempio se `prossimoServito` è 3 (ovvero 2 è attualmente allo sportello) e il `numero` dell'utente è 5 con una fila dove: i minuti dell'utente 2 sono stimati a 5, quelli di 3 sono 10, quelli di 4 sono 2, e quelli di 5 sono 3; il metodo deve restituire 12. Nel caso in cui `numero` corrisponde ad un utente non in fila (ovvero un numero già chiamato o che non è stato ancora ritirato) il metodo deve restituire -1.

2 JML

Scrivere in JML il seguente invariante:

1. `prossimoInFila` ha un valore tra 0 e 200 inclusi e `prossimoServito` non è mai strettamente superiore a `prossimoInFila`. **[pt. 0.75]**

Scrivere in JML la seguente postcondizione al costruttore:

2. non ci sono utenti in fila, verificando esclusivamente il contenuto dell'array `fila[]`. **[pt. 0.50]**

Scrivere in JML la seguente precondizione al metodo `ritiroNumero(int minuti)`:

3. i minuti di attesa di un utente sono compresi tra 1 e 15 inclusi. **[pt. 0.25]**

Scrivere in JML le seguenti postcondizioni al metodo `ritiroNumero(int minuti)`:

4. tutti gli elementi dell'array `fila[]` di indice strettamente inferiore a `prossimoInFila` hanno valore strettamente maggiori di 0. **[pt. 0,75]**

5. almeno un utente è stato inserito in fila, verificando esclusivamente il contenuto dell'array `fila[]`. **[pt. 0,75]**

6. nell'array `fila[]` non c'è mai uno valore a 0 prima di uno strettamente maggiore di 0. **[pt. 1.25]**

Scrivere in JML la seguente postcondizione al metodo `stimaMinuti(int numero)`:

7. se `numero` è già stato chiamato il metodo restituisce -1. **[pt. 0,75]**

3 Junit

In JUnit, scrivere i seguenti casi di test nella classe `FilaTest`.

1. Per il metodo `ritiroNumero(int minuti)`, scrivere un caso di test che mostri che, se si prova a ritirate un numero quando la fila è piena il numero non viene fornito (ovvero si ottiene -1); **[pt. 1]**

2. Scrivere un caso di test che mostri che è possibile riempire completamente la fila in modo che l'ultimo utente debba aspettare esattamente 1000 minuti. **[pt. 1]**

4 Copertura

Scrivere in JUnit una test suite per il metodo `ritiroNumero` che soddisfi la copertura delle *decisioni*. Creare una classe `FilaCoperturaDecisioni` e commentare in modo opportuno i singoli casi di test. (Si ricorda che il metodo ha successo quando restituisce un valore diverso da -1.) Per testare la copertura mettere la variabile `prossimoInFila` come public. **[pt. 1]**

Totale punti = 2 + 5 + 2 + 1= 10