Verifica: esercitazione

Vincenzo Gervasi, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

Rebu1: euristica di selezione dell'autista

```
public Autista seleziona(Set<Autisti> disponibili, Richiesta r){
      int score=1000, d=0;
      Autista prescelto=null;
          Location aLoc=null, rLoc=r.getLocation();
      for (Autista a : disponibili) {
            aLoc=a.getLocation();  d=aLoc.distance(rLoc);
            if (d<score) {
                  score=d;
                  prescelto=a;
            }
      }
      return a;
}
```

Si definiscano:

1. Un insieme minimo di valori di input che garantisca una copertura del 100% dei comandi.
2. Una partizione del dominio degli argomenti del metodo seleziona: si ripartisca in classi di equivalenza rispetto al criterio di copertura delle decisioni;
3. Si svolga infine una ispezione strutturata del codice, definendo una checklist che includa la verifica del trattamento dei null. Si evidenzia qualche problema nel codice dato sopra?

Rebu 2: emissioni

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.

Rebu 2, emissioni: Si consideri la seguente implementazione

```
public void calcolaGiriMotore (double valoreSensoreAcceleratore){
      int numeroGiri, numeroGiriFake;
      \\ per una opportuna funzione f
      numeroGiri = f(valoreSensoreAcceleratore);
      if(inMovimento(ruoteMotrici)&!inMovimento(ruoteNonMotrici))
            numeroGiriFake = (int) 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;

Rebu 3

Si esegue un test black box del metodo calcolaSpesaSettimanale, che, dato un array di viaggi effettuati da un profilo business in una settimana, calcola la spesa totale. Si provano i seguenti casi di test, con il risultato riportato accanto a ciascuno. Indichiamo i viaggi con la notazione (…, *costo*), in cui i puntini astraggono dettagli non significativi.

```
< [ ], 0, _ >, 0
< [(…, 0)], 0, _ >, 0
< [(…, 5)], 5, _ >, 0
< [(…, 5),(…, 16), (…,22)], 43, _ >, 38
```

Si riesce, da questi risultati, a ipotizzare eventuali difetti nel codice?
Si definisca un elemento di checklist per cercare altre occorrenze di difetti analoghi che possono essere presenti nel codice.

Rebu4: calcolaPartenzaPerAeroporto

Il metodo calcolaPartenzaPerAeroporto calcola l'orario di inizio di una corsa, conoscendo, l'anticipo, il tempo di percorrenza (che si assume qui indipendente dall'orario del volo), e il volo. Inoltre, restituisce un nuovo orario di partenza in caso di scostamenti superiori ai 15 minuti rispetto a un orario precedentemente calcolato.

Si consideri il seguente frammento di codice, dove il valore del parametro orarioPrevistoCorsa è -1 quando non è ancora stato calcolato un orario di partenza, diverso altrimenti:

Rebu4: calcolaPartenzaPerAeroporto

```
private int calcolaPartenzaPerAeroporto(int orarioPrevistoCorsa, int anticipo, int tempoPrecorrenza,
Volo v){
        int o = v.getOrarioPartenza();
        int d = o - anticipo - tempoPercorrenza ;
        int t = d - now();

        if (t > 120 || orarioPrevistoCorsa == -1)
                return d;
        if (t<0)
                return -5;
        int r = d – orarioPrevistoCorsa;
        if (Math.abs(r)>15)
                return d;
        else
                return orarioPrevistoCorsa;
}
```

Dare il diagramma di flusso del metodo e un insieme minimo di casi di test per avere copertura delle decisioni e copertura delle condizioni.

Rebu5

Si consideri il seguente frammento di codice, che ha lo scopo di controllare se un intervallo temporale A sia interamente contenuto all'interno di un intervallo temporale B (usato dal sistema REBU per controllare se una richiesta di disponibilità di auto condivisibile è compatibile con l'orario di disponibilità di una particolare auto messa in condivisione):

```
public boolean inside(Timespan a, Timespan b) {
                    if (a.start < b.start)  return false;
                    if (a.end > b.end)     return false;
                    if ((a.start >= b.start) && (a.end <= b.end))          return true;
                    return false;
}
```

Si adotti un atteggiamento di *defensive programming*, ovvero ci si proponga di realizzare una suite di test che consenta di verificare il funzionamento del metodo inside() senza fare assunzioni sulla correttezza dei parametri.
Oltre a dare la lista di casi di test, si commenti su quale criterio o insieme di criteri sono stati usati per generarla, e si propongano eventuali correzioni al codice (derivanti dai risultati del test).

CellEx

Una delle funzioni ausiliarie del sistema CellEx è di fornire informazioni statistiche sull'andamento degli esami. In particolare, il sistema deve fornire informazioni sul numero d'esami mediamente sostenuti ogni giorno, per corso di laurea, facoltà, e per tutta l'università. A tale scopo, si prevede l'utilizzo di una funzione numeroMedioEsami che, dato un vettore di numeri d'esame, ne restituisce la media, arrotondata all'intero superiore. Per verificare la funzione si prevede un test a scatola nera.

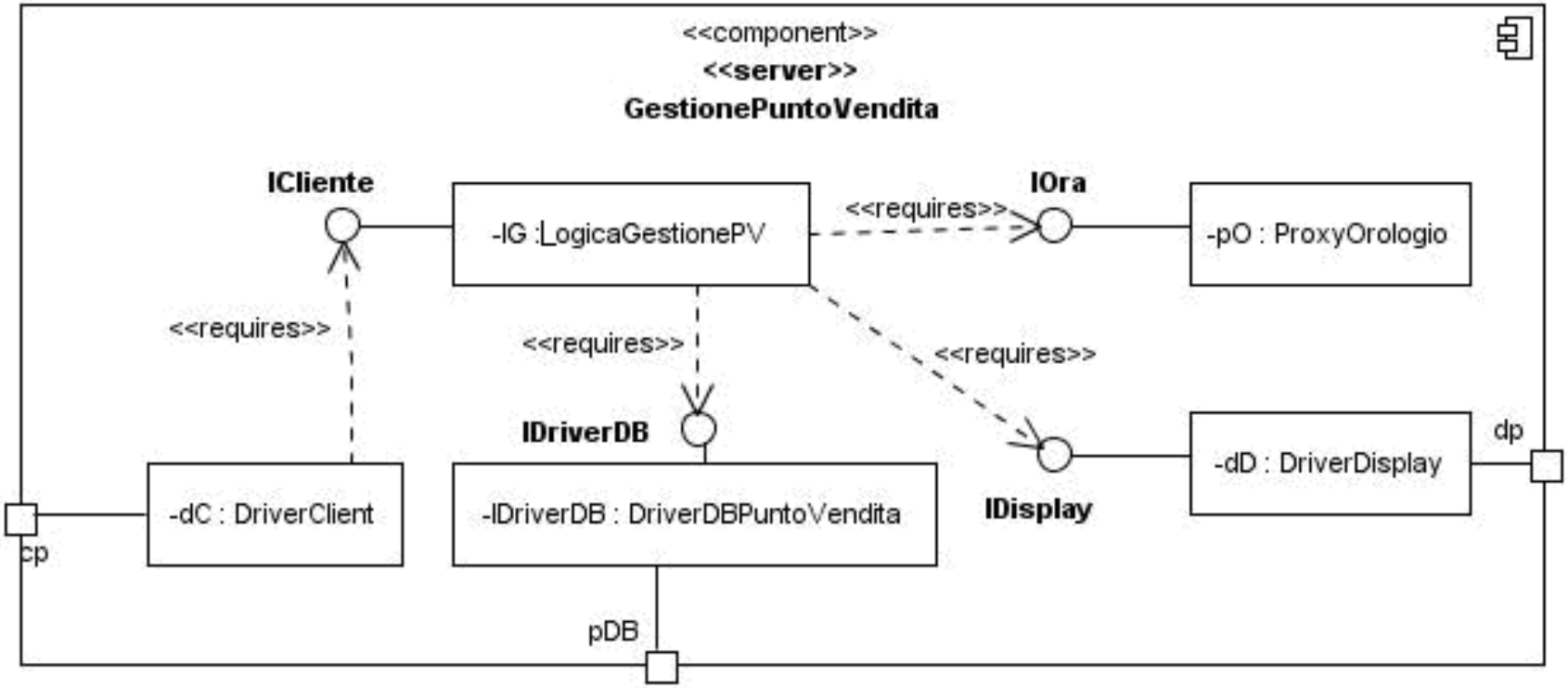
Domanda. Fornire cinque casi di prova per la funzione numeroMedioEsami, giustificando per ciascuno la ragion d'essere.

CellEx: risposta

Casi di prova		Giustificazione
Input: valori	Output: media	
[ ]	0	**Caso limite: vettore vuoto**
[1]	1	**Caso limite: un solo elemento**
[1,1]	1	**Caso speciale: tutti uguali**
[1,2]	2	**Verifica arrotondamento**
[4,1,2]	**3**	**Caso generico**

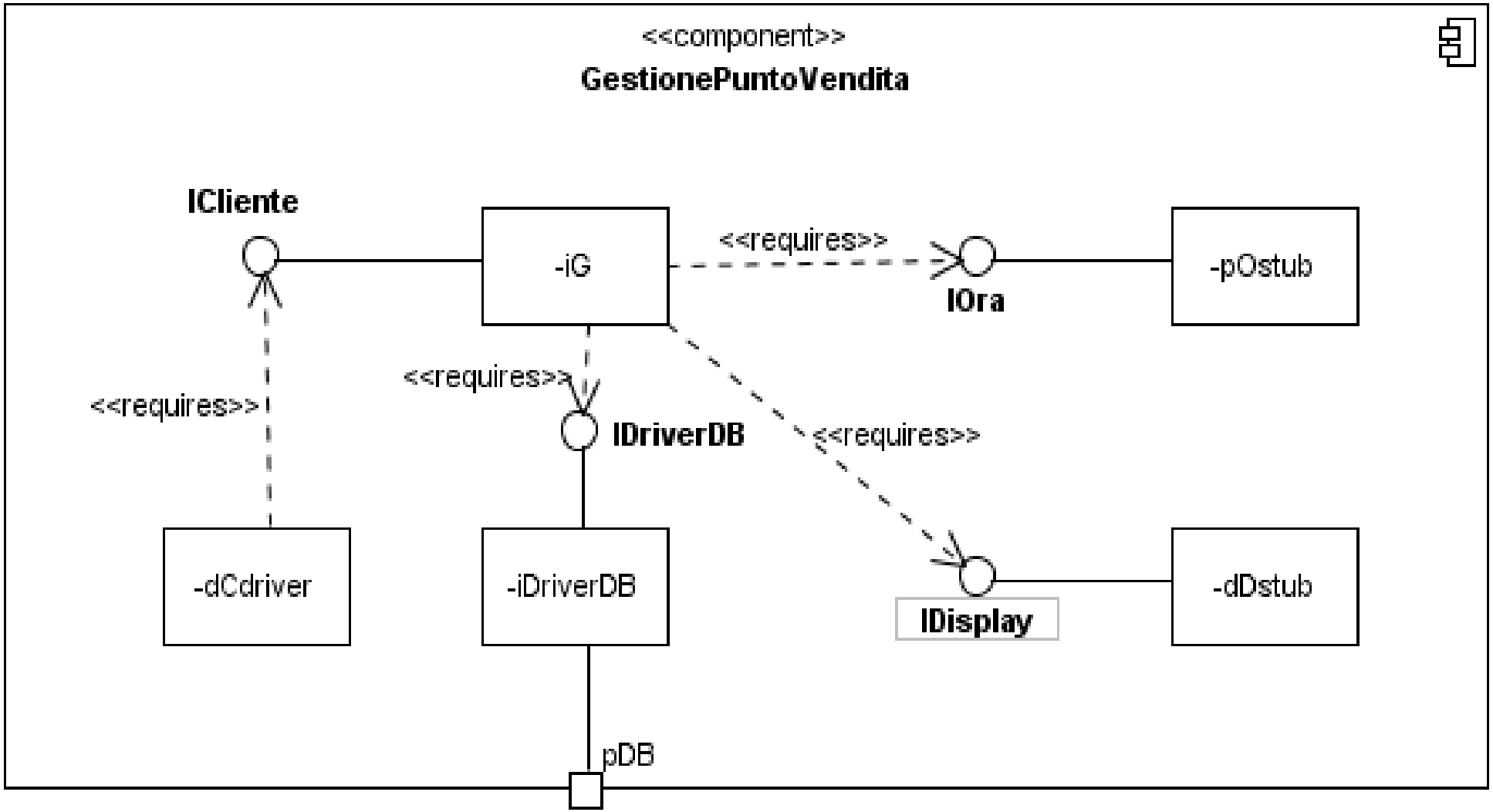
Stub e Driver

- Dato il seguente diagramma di struttura composita



descrivere, con un diagramma di struttura composita, l'ambiente di verifica (stub e driver) di LogicaGestionePV. Si assuma di avere già testato il database, e quindi poterlo utilizzare per il test.

Stub e Driver: risposta



Borghi (Causa-effetto)

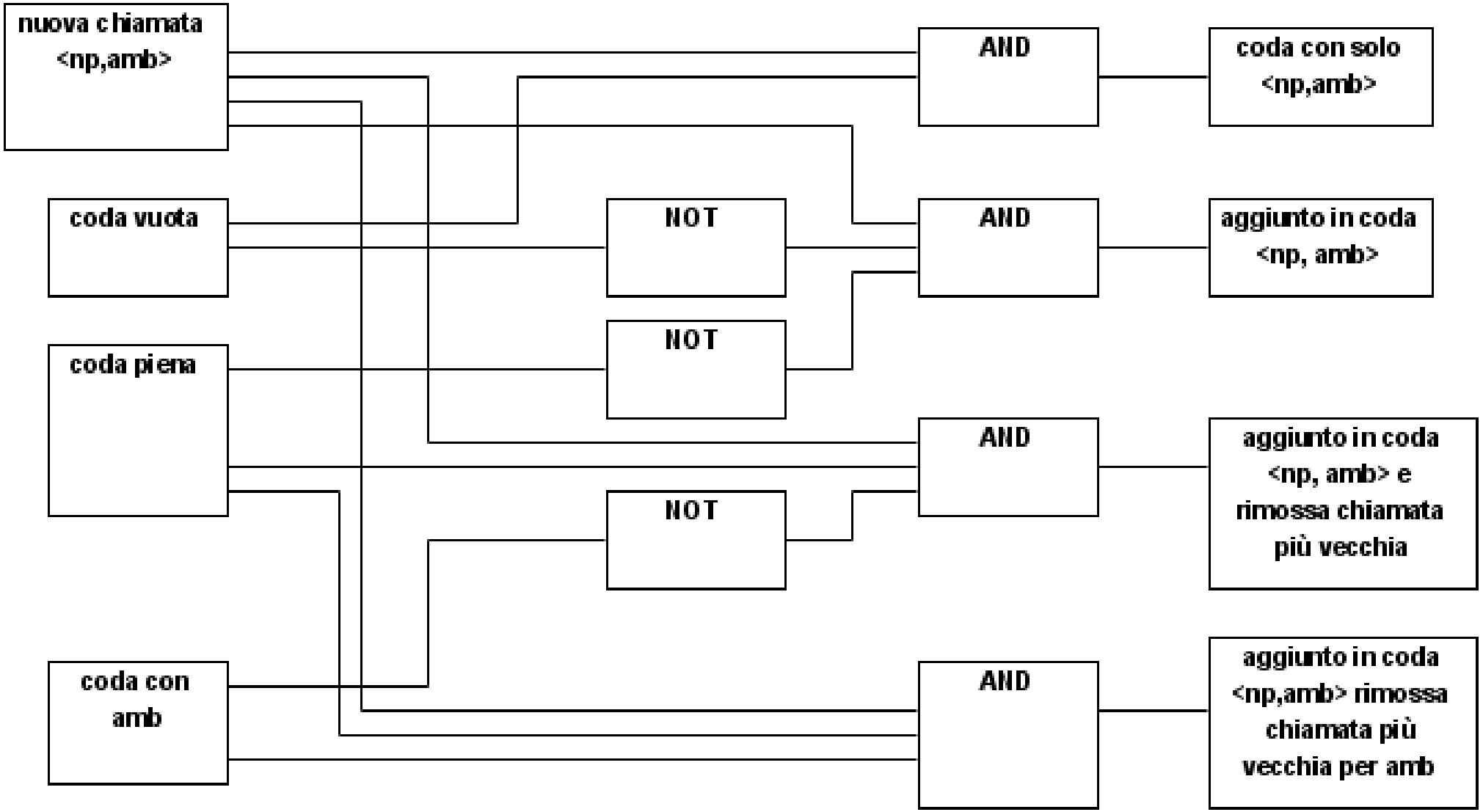
Il sottosistema di supporto alle attività ambulatoriali deve permettere al personale medico di:

1. ..
2. chiamare i pazienti in attesa aggiornando i visori posti nelle sale di aspetto;

Il visore è in grado di mostrare al più 20 chiamate, rappresentate da coppie <np, amb> (numero di prenotazione, ambulatorio). Ogni nuova richiesta di visualizzazione viene inserita in coda alle precedenti. Se lo spazio a disposizione è esaurito, viene cancellata una chiamata: la più vecchia relativa allo stesso ambulatorio, se ne esiste almeno una, o la più vecchia in assoluto, altrimenti.

Dare un diagramma di causa-effetto per la progettazione dei casi di test per il funzionamento del visore, considerando le seguenti cause: nuova chiamata <np,amb>, coda vuota, coda piena, coda contenente amb.

Borghi: risposta



Rebu: esecuzione simbolica

```
1  private int calcolaPartenzaPerAeroporto(Prenotazione p)
2  {
3          int s;
4          int a = p.getAnticipo();
5          Volo v = p.getVolo();
6          Location l1 = p.getIndirizzoCasa();
7          Location l2 = v.getIndirizzoAeroportoPartenza();
8          int p = tempoPercorrenza(l1,l2);
9          int o = v.getOrarioPartenza();
10         int d = o - a - p;
11         int t = d - now();
12         if (t <= 120) {
13                 int r = v.getRitardo();
14                 if (r>15)
15                         s = d + r;
16                 else
17                         s = d;
18         }
19         return s;
20 }
```

Considerando che gli intervalli temporali sono espressi in minuti, e gli orari assolut icome numero di secondi trascorsi da un momento fissato (l'epoch di sistema), la funzione restituisce l'ora di partenza
prevista verso l'aeroporto (in caso di prenotazione l egata a un volo in partenza).

1. si mostrino gli stati (calcolati tramite esecuzione simbolica) in corrispondenza delle righe 10,11, 15, 17, 19;
2. Si determini sotto quali condizioni il valore di ritorno può essere successivo all'ora di partenza prevista per il volo (ovvero, s > o alla riga 19);
3. si determini sotto quali condizioni il valore di ritorno può essere precedente a quello calcolato inizialmente (ovvero, s < d alla riga 19).

In particolare, le risposte alle domande 2. e 3. suggeriscono qualche postcondizione da definire sui risultati dei vari metodi chiamati alle righe 4-9, 11, 13? Se ne faccia qualche esempio.

CicloPi: black box

CicloPi è gratuito per le corse di durata inferiore ai 30 minuti, anche più volte al giorno. Se l'utilizzo supera i 30 minuti consecutivi, sarà applicata la tariffazione relativa alla propria formula di abbonamento scalando l'importo dal credito presente sulla tessera. Il costo è di €0,90 la seconda mezz'ora (o frazione), €1,50 la terza, €2 dalla quarta mezz'ora in poi.
Rappresentando i valori della classe Data con gg/mm/aa – hh:mm, si è definito il metodo

double calcolaCostoBiciNonDanneggiata(Data dataInizio, Data dataFine)

calcola il costo di utilizzo di una bicicletta al momento della riconsegna.

Dare un insieme di casi di test progettati secondo i seguenti criteri a scatola chiusa: statistico, partizione dei dati di ingresso, frontiera.

CicloPi: risposta

oraInizio	oraFine	output	ragione
18/05/16-08:00	18/05/16-08:00	0.00	Caso limite, possibile per ripensamenti (o sella alta/bassa)
18/05/16-08:00	18/05/16-08:10	0.00	Partizione 1, statisticamente più probabile
18/05/16-08:00	18/05/16-08:15	0.00	Partizione 1, statisticamente più probabile
18/05/16-08:00	18/05/16-08:29	0.00	Partizione 1, statisticamente più probabile
18/05/16-08:00	18/05/16-08:30	0.90	Frontiera
18/05/16-08:00	18/05/16-08:45	0.90	Partizione 2
18/05/16-08:00	18/05/16-09:00	2.40	Frontiera
18/05/16-08:00	18/05/16-09:45	2.40	Partizione 3
18/05/16-08:00	18/05/16-09:30	4.40	Frontiera
18/05/16-08:00	18/05/16-09:45	4.40	Partizione 4
18/05/16-08:00	19/05/16-8:00	94:40	Frontiera 24 ore
18/05/16-08:00	19/05/16-8::15	94:40	Partizione 24 ore e rotti

Stub e Criteri strutturali

Il pedaggio si calcola considerando: la tariffa unitaria a chilometro, il tipo di veicolo utilizzato (5 classi), le caratteristiche dei tratti autostradali percorsi (di pianura o di montagna).
Si supponga il calcolo sia fatto usando i metodi così specificati:

/*dati i caselli di ingresso e di uscita, restituisce il numero di km di pianura e il numero di quelli di montagna*/
int[ ] calcolaChilometri(a String, b String)

/*dati i caselli di ingresso e di uscita e la classe del veicolo, ottiene il numero di Km percorsi e calcola il pedaggio */
double calcolaPedaggio(a String, b String, c Classe)

Per quale dei due metodi dati sopra potrebbe essere utile creare uno stub, nella verifica del calcolo del pedaggio? Definire un semplice stub, che permetta la ripetibilità dei test, e non sia banale (vari i risultati in funzione degli argomenti).

Stub e Criteri strutturali: risposta

Notando che calcolaPedaggio deve invocare calcolaChilometri, e che la realizzazione di questo metodo richiede l'accesso al DBrete, conviene testare calcolaPedaggio con uno stub per calcolaChilometri.
Per permettere la ripetibilità dei test non si può usare un generatore di numeri pseudo-casuali. La soluzione che segue conserva la proprietà commutativa di calcolaChilometri (andando da A a B si fanno gli stessi chilometri che andando da B a A) e può produrre anche risultati estremi (tratto di montagna o di pianura lungo zero):

```
int[ ] calcolaChilometri(a String, b String){
      int[] coppia = {0,0};
      int mx = max(a.length(),b.length());
      int mn = min(a.length(),b.length());
      coppia[0]= mx - mn ;
      coppia[1]= (2*mn >= mx ? 2*mn-mx : mn) ;
      return coppia;
}
```

Albergo dei Fiori

Il seguente metodo determina la durata del più lungo periodo di occupazione di una stanza in un periodo dato.

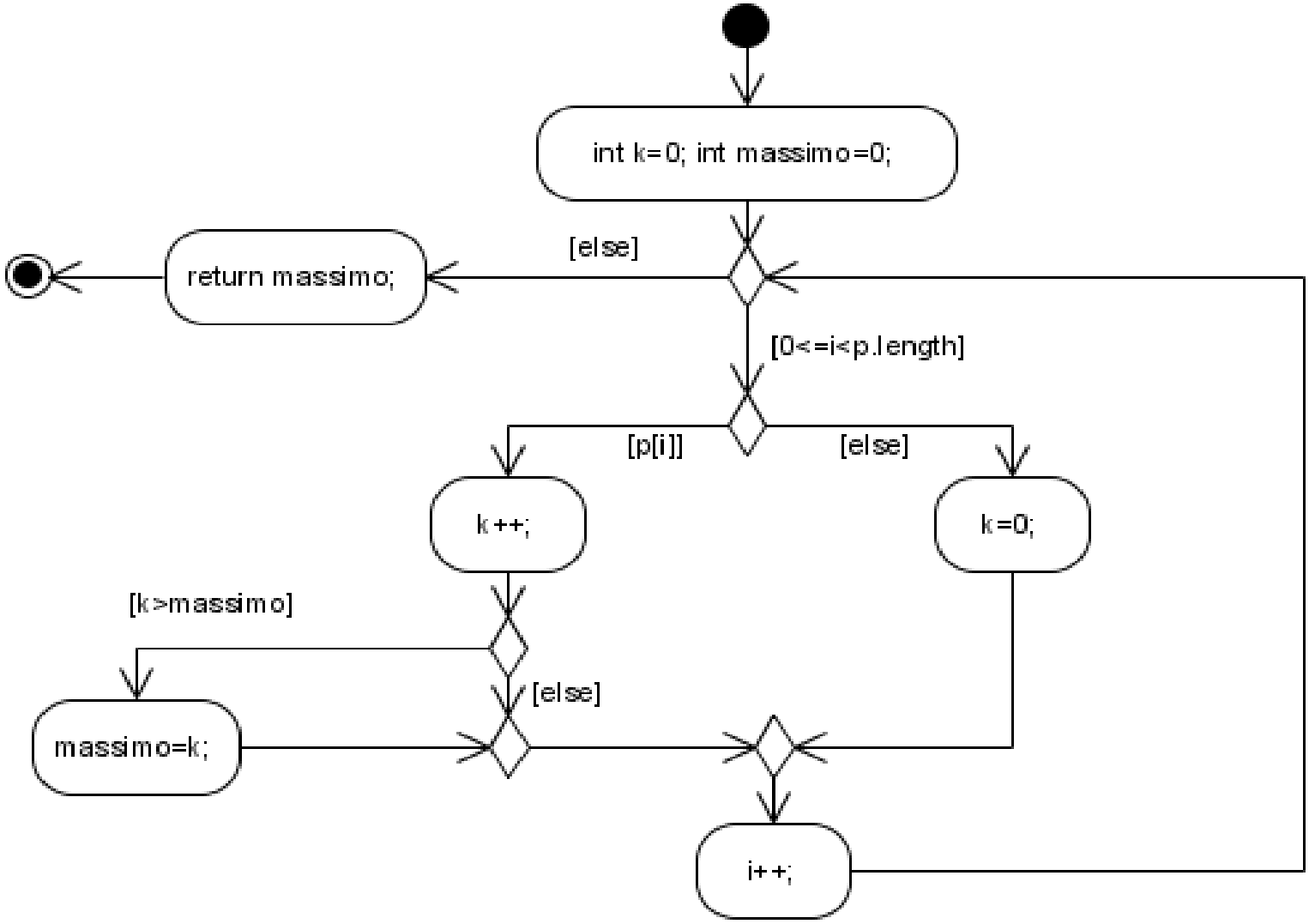
```
public int massimoPeriodo (boolean [] p) {
    int k = 0;
    int massimo = 0;
    for (int i = 0; i < p.length; i++) {
        if (p[i]) {
            k++;
            if (k > massimo) {
                massimo = k;
            }
        } else {
            k = 0;
        }
    }
    return massimo;
}
```

Domanda.

a) Disegnare il grafo di flusso (o grafo di controllo) del metodo, usando un diagramma di attività

b) Dare un insieme di cardinalità minima di casi di prova per la copertura delle decisioni.

Albergo dei Fiori: risposta a)



Albergo dei Fiori: risposta b)

Un insieme minimale di casi di prova che soddisfa la copertura richiesta è il seguente:

input	output
T F T	1