Verifica: esercitazione

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 (dare un rappresentante per classe e le proof obbligations);
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?

Rebu1: RISPOSTA Definire un insieme minimo di valori di input che garantisca una copertura del 100% dei comandi.

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

Set<Autisti> disponibili	Richiesta r
{Andrea| Andrea.location=stazione }	r1 | r1.location= piazzaVittorio

Rebu1: RISPOSTA Definire una partizione del dominio degli argomenti del metodo seleziona: si ripartisca in classi di equivalenza rispetto al criterio di copertura delle decisioni

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

Set<Autisti> disponibili	Richiesta r	Decisioni coperte (etichettate con le linee)
{ }	r1 | r1.location= p.zaVittorio	1 → esce subito dal ciclo
{Andrea| Andrea.location=stazione}	r1 | r1.location= p.zaVittorio	1→ 2,3 →4,5 ,1 →esce
{Andrea, Luigi | Andrea.location=stazione Luigi.location=torrePisa}	r1 | r1.location= p.zaVittorio	1→ 2,3 →4,5 , 1→2,3 → 1 → esce

Le soluzione chiedeva le classi: { }, {un autista solo oppure n autisti ordinati dal più lontano al più vicino}, {due o più autisti ordinati dal più vicino al più lontano}

Rebu1: RISPOSTA 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?

```
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;
}
```

- Un check è controllare il valore che viene assegnato alla variabile restituita→ difetto, variabile locale al ciclo for (se ne sarebbe comunque accorto il compilatore)
- Anche fosse stato `return prescelto` la variabile sarebbe potuta essere `null` (se `disponibili` è vuoto o se tutte le distanze calcolate sono >1000)

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.

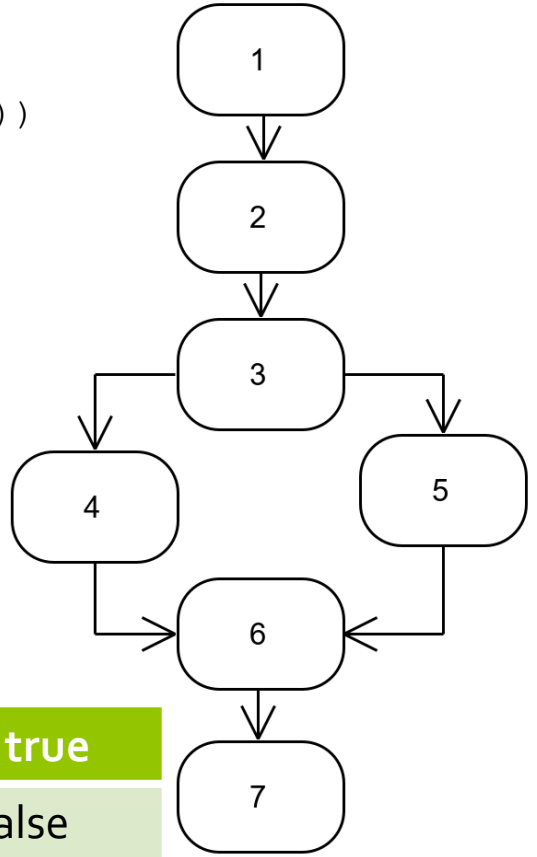
Rebu 2, emissioni: Si consideri la seguente implementazione

```
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 2, emissioni: RISPOSTA 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;

```
public void calcolaGiriMotore (double valoreSensoreAcceleratore){
1.      int numeroGiri, numeroGiriFake;
        \\ per una opportuna funzione f
2.      numeroGiri = f(valoreSensoreAcceleratore);
3.      if(inMovimento(ruoteMotrici)&!inMovimento(ruoteNonMotrici))
4.              numeroGiriFake = (int) 0.7 * numeroGiri;
5.      else numeroGiriFake = numeroGiri;
6.      … // scrivi numeroGiri nel file di log;
7.      … // invia numeroGiriFake al motore;
}
```



Si noti intanto che il valore del paramentro di input è completamente ininfluente per la copertura delle decisioni

inMovimento(ruoteMotrici) = true	inMovimento(ruoteNonMotrici) = true
inMovimento(ruoteMotrici) = true	inMovimento(ruoteNonMotrici) = false

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.

Rebu 3 RiSPOSTA

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?

→ **non viene considerato il primo elemento dell'array**

Si definisca un elemento di checklist per cercare altre occorrenze di difetti analoghi che possono essere presenti nel codice.

→ **difetti di tipo off-by-one**

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

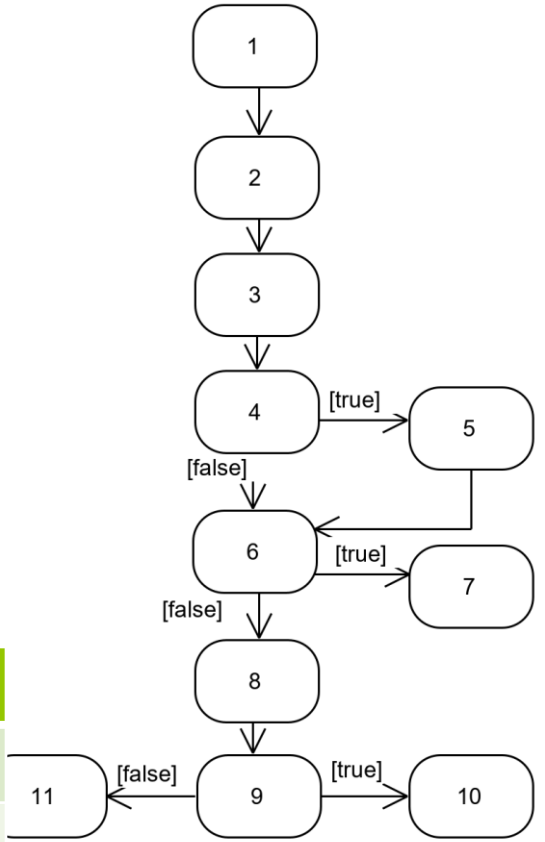
```
private int calcolaPartenzaPerAeroporto(int orarioPrevistoCorsa, int anticipo, int tempoPrecorrenza,
Volo v){
        int o = v.getOrarioPartenza();
        int d = o - anticipo - tempoPercorrenza ;
        int t = d - now();

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

(a) Dare il diagramma di flusso del metodo e un insieme minimo di casi di test per avere copertura delle decisioni e copertura delle condizioni (multiple condition coverage).
(b) Quale di questi parametri, in un testing combinatorio, sarebbe un buon candidato per essere [single] ? Perche?

Rebu4: RISPOSTA (completare) Dare il diagramma di flusso del metodo e un insieme minimo di casi di test per avere copertura delle decisioni……

```
private int calcolaPartenzaPerAeroporto(int orarioPrevistoCorsa, int
anticipo, int tempoPrecorrenza, Volo v){
1.      int o = v.getOrarioPartenza();
2.      int d = o - anticipo - tempoPercorrenza ;
3.      int t = d - now();
4.      if (t > 120 || orarioPrevistoCorsa == -1)
5.              return d;
6.      if (t<0)
7.              return -5;
8.      int r = d – orarioPrevistoCorsa;
9.      if (Math.abs(r)>15)
10.             return d;
11.     else    return orarioPrevistoCorsa;
}
```



orarioPrevistoCorsa	anticipo	tempoPrecorrenza	v.orarioP	now	archi
320	30	10	400	100	4->5
320	30	10	400	300	4->6; 6->8; 9->10
…	…	…..	…..	….	……

Rebu4: RISPOSTA (completare) Dare il diagramma di flusso del metodo e un insieme minimo di casi di test per avere copertura …. e delle condizioni (oltre a quelli visti)

```
private int calcolaPartenzaPerAeroporto(int orarioPrevistoCorsa,
int anticipo, int tempoPrecorrenza, Volo v){
1.      int o = v.getOrarioPartenza();
2.      int d = o - anticipo - tempoPercorrenza ;
3.      int t = d - now();
4.      if (t > 120 || orarioPrevistoCorsa == -1)
5.              return d;
6.      if (t<o)
7.              return -5;
8.      int r = d – orarioPrevistoCorsa;
9.      if (Math.abs(r)>15)
10.             return d;
11.     else    return orarioPrevistoCorsa;
}
```

I due casi di test coprono le decisioni. Per la multiple condition coverage aggiungere un caso di test che contenga t<=120

orarioPrevistoCorsa	anticipo	tempoPrecorrenza	v.orarioP	now	CONDIZIONI
-1	30	10	400	100	t>120 true e orarioPrevistoCorsa ==-1
320	30	10	400	300	t>120 true e orarioPrevistoCorsa !=-1

Rebu4: RISPOSTA Quale di questi parametri, in un testing combinatorio, sarebbe un buon candidato per essere [single] ? Perche?

```
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;
}
```

anticipo e tempoPrecorrenza perché si usano solo in d = o - anticipo - tempoPercorrenza e per avere diversi valori di d basta cambiare o.

In generale, si possono dichiarare [single] 2 tra (v, anticipo, tempoPercorrenza)

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).

Rebu5 SOLUZIONE

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). **(a.start >a.end, lo stesso per b) cambiare i codice per fare un controllo**

Test combinatorio

Si consideri la seguente proof obligation:

C1

- V_1
- V_2

C2

- V_3
- V_4

C3

- V_5
- V6
- V_7

Volendo fare un test esaustivo con tutte le combinazioni, quale sarebbe la dimensione della test suite ?

Test combinatorio RISPOSTA

Si consideri la seguente proof obligation:

C1

- V_1
- V_2

C2

- V_3
- V_4

C3

- V_5
- V6
- V_7

Volendo fare un test esaustivo con tutte le combinazioni, quale sarebbe la dimensione della test suite

Risposta: 2x2x3=12

Test combinatorio (Pairwise)

Si consideri la seguente proof obligation:

C1

- V_1
- V_2

C2

- V_3
- V_4

C3

- V_5
- V6
- V_7

Volendo considerare le sole combinazioni a coppie, quale sarebbe la test suite?

Test combinatorio (Pairwise) RISPOSTA

Si consideri la seguente proof obligation :

C1

- V_1
- V_2

C2

- V_3
- V_4

C3

- V_5
- V6
- V_7

Volendo considerare le sole combinazioni a coppie, quale sarebbe la test suite?→

Risposta: (6 casi di test)

V_5	V_1	V_4
V_5	V_2	V_3
V6	V_1	V_3
V6	V_2	V_4
V_7	V_1	V_3
V_7	V_2	V_4

Test combinatorio (if property)

Si ora consideri la seguente proof obligation :

C1

- V1 [property P1]
- V2 [property P2]

C2

- V3 [property P3]
- V4 [property P4]

C3

- V5 [if P1]
- V6 [if P4]
- V7

E ora, quale sarebbe la test suite?

Test combinatorio (if property) RISPOSTA

Si ora consideri la seguente specifica:

C1

- V_1 [property P_1]
- V_2 [property P_2]

C2

- V_3 [property P_3]
- V_4 [property P_4]

C3

- V_5 [if P_1]
- V6 [if P_4]
- V_7

E ora, quale sarebbe la test suite?

Risposta: (8 casi di test)

V_5	V_1	V_3
V_5	V_1	V_4
V6	V_1	V_4
V6	V_2	V_4
V_7	V_1	V_3
V_7	V_2	V_4
V_7	V_1	V_4
V_7	V_2	V_3

Test combinatorio (if property e error)

Si ora consideri la seguente specifica:

C1

- V1 [property P1]
- V2 [property P2]

C2

- V3 [property P3]
- V4 [property P4]

C3

- V5 [if P1]
- V6 [if P4]
- V7 **[error]**

E ora, quale sarebbe la test suite?

Test combinatorio (if property e error) RISPOSTA

Si ora consideri la seguente specifica:

C1

- V1 [property P1]
- V2 [property P2]

C2

- V3 [property P3]
- V4 [property P4]

C3

- V5 [if P1]
- V6 [if P4]
- V7 **[error]**

E ora, quale sarebbe la test suite?

Risposta: (5 casi di test)

V5	V1	V3
V5	V1	V4
V6	V1	V4
V6	V2	V4
V7	V1	V3

Test mutazionale

```
 /* edit1( s1, s2 ) returns TRUE iff s1 can be transformed to s2 by inserting, deleting,
or substituting a single character, or  by a no-op (i.e., if they are already equal).*/
...
6 int edit1( char *s1, char *s2) {
7 if (*s1 == '/0') {
8 if (*s2 == '/0?) return TRUE;
9 /* Try inserting a character in s1 or deleting in s2 */
10 if ( *(s2 + 1) == '/0') return TRUE;
11 return FALSE;
12 }
13 if (*s2 == '/0') { /* Only match is by deleting last char from s1 */
14 if (*(s1 + 1) == '/'0) return TRUE;
15 return FALSE;
16 }
17 /* Now we know that neither string is empty */
18 if (*s1 == *s2) {
19 return edit1(s1 +1, s2 +1);
20 }
21
22 /* Mismatch; only dist 1 possibilities are identical strings after
23 * inserting, deleting, or substituting character
24 */
25
26 /* Substitution: We "look past" the mismatched character */
27 if (strcmp(s1+1, s2+1) == 0) return TRUE;
28 /* Deletion: look past character in s1 */
29 if (strcmp(s1+1, s2) == 0) return TRUE;
30 /* Insertion: look past character in s2 */
31 if (strcmp(s1, s2+1) == 0) return TRUE;
32 return FALSE;
33 }
```

Si consideri la mutazione ottenuta sostituendo, alla linea 27, s1 +1 con s1 + 0.

Dare un caso di test che uccide il mutante

Definire:
Un mutante invalido;
Un mutante valido ma inutile (che fallisce per quasi ogni caso di test);
Un mutante valido ma equivalente;

PisaMover 1

Si consideri il seguente codice, che implementa (?) la funzionalità "tentativi multipli di chiusura delle porte in presenza di ostacoli". La specifica del metodo prevede che ensureClosed(*d*) restituisca **true** se e solo se la porta *d* è verificata chiusa al momento del ritorno dal metodo.

```
boolean ensureClosed(Door d){
        for (int attmpt=0; attmpt<3; attmpt++) {
                if (d.isClosed())  return true;
                d.send(CMD_CLOSE);
                await(2000);
                if ((!d.isClosed()) || d.hasFault()) {
                        d.send(CMD_OPEN); await(2000);
                } else return d.isClosed();}
        return false;}
```

1. si produca una test suite minimale che garantisca il 100% di copertura dei comandi;
2. si produca una test suite minimale che garantisca il 100% di copertura delle condizioni;
3. i test identificati ai punti precedenti evidenziano uno o più difetti nel codice? Se si, indicare quali. Se no, indicare se si ritiene che siano presenti dei difetti, e quale o quali test case li metterebbe in evidenza.

Pisamover 3

Un Biglietto ha, tra i suoi attributi, l'ora di uscita prevista, in formato LocalDateTime:
A date-time without a time-zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30.

Il metodo
int calcolaPrezzo(Biglietto b, LocalDateTime now)
di una qualche classe del vostro sistema restituisce il prezzo da pagare per saldare il parcheggio.

Definire (dandone il codice) un driver di test che sia anche factory di oggetti di tipo Biglietto per verificare la correttezza del metodo calcolaPrezzo(), garantendo una opportuna copertura secondo i criteri funzionali. Commentare le scelte fatte.

PisaMover 4

Si consideri il metodo

public int maxSpeed(SensorReading[] sens, Direction d)

che, dato un array di letture dei sensori (si assuma che ciascun sensore possa riconoscere i vagoni, indicare la presenza o meno di un vagone in corrispondenza della posizione del sensore, e che l'ordine dei sensori nell'array coincida con l'ordine in cui sono installati lungo il binario), e una direzione di marcia, restituisca la velocità massima che il convoglio dovrebbe avere in quel momento.

Si definisca una test suite contenente al massimo 5 test case, cercando di ottenere la massima copertura possibile del dominio dei parametri di input, e si fornisca una stima del grado di copertura così raggiunto.

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.

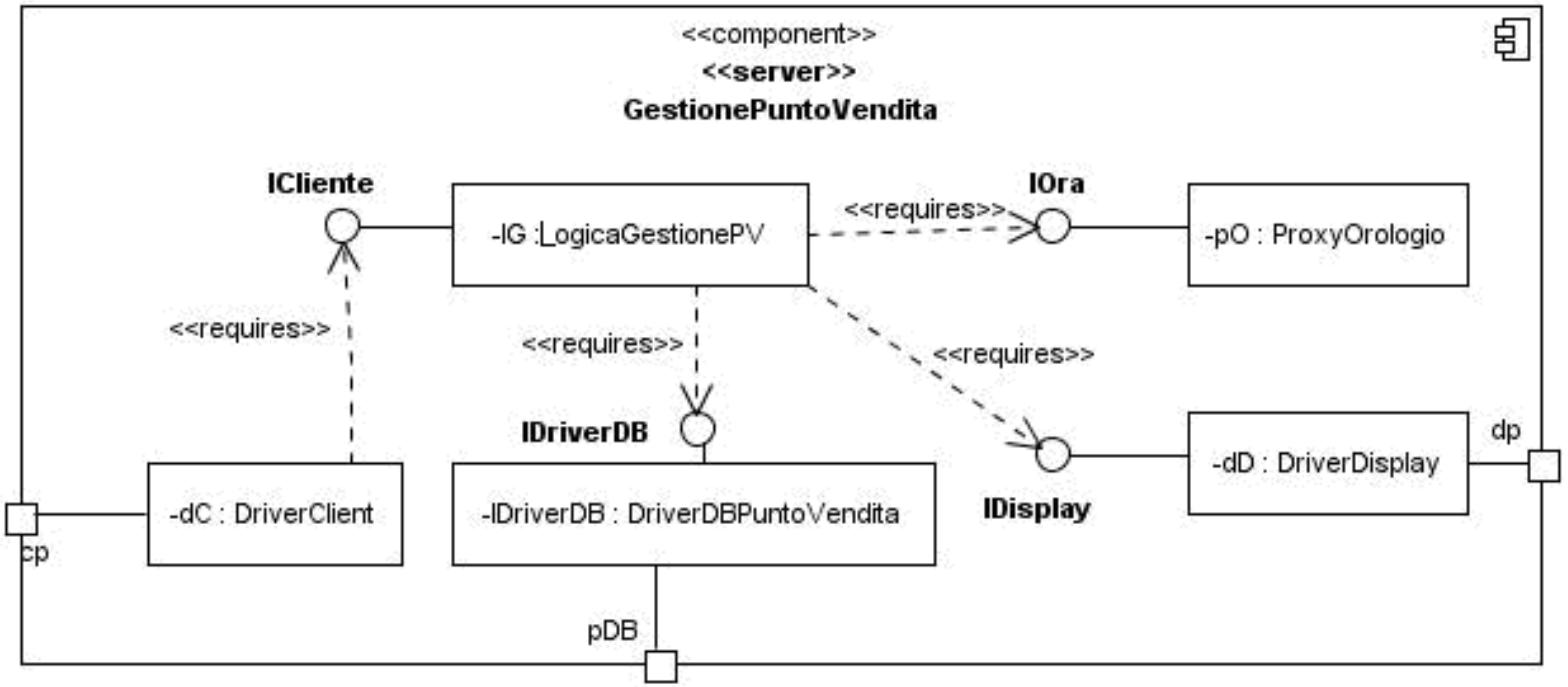
Domanda. Fornire cinque casi di prova per la funzione numeroMedioEsami, giustificando per ciascuno la ragion d'essere.

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**

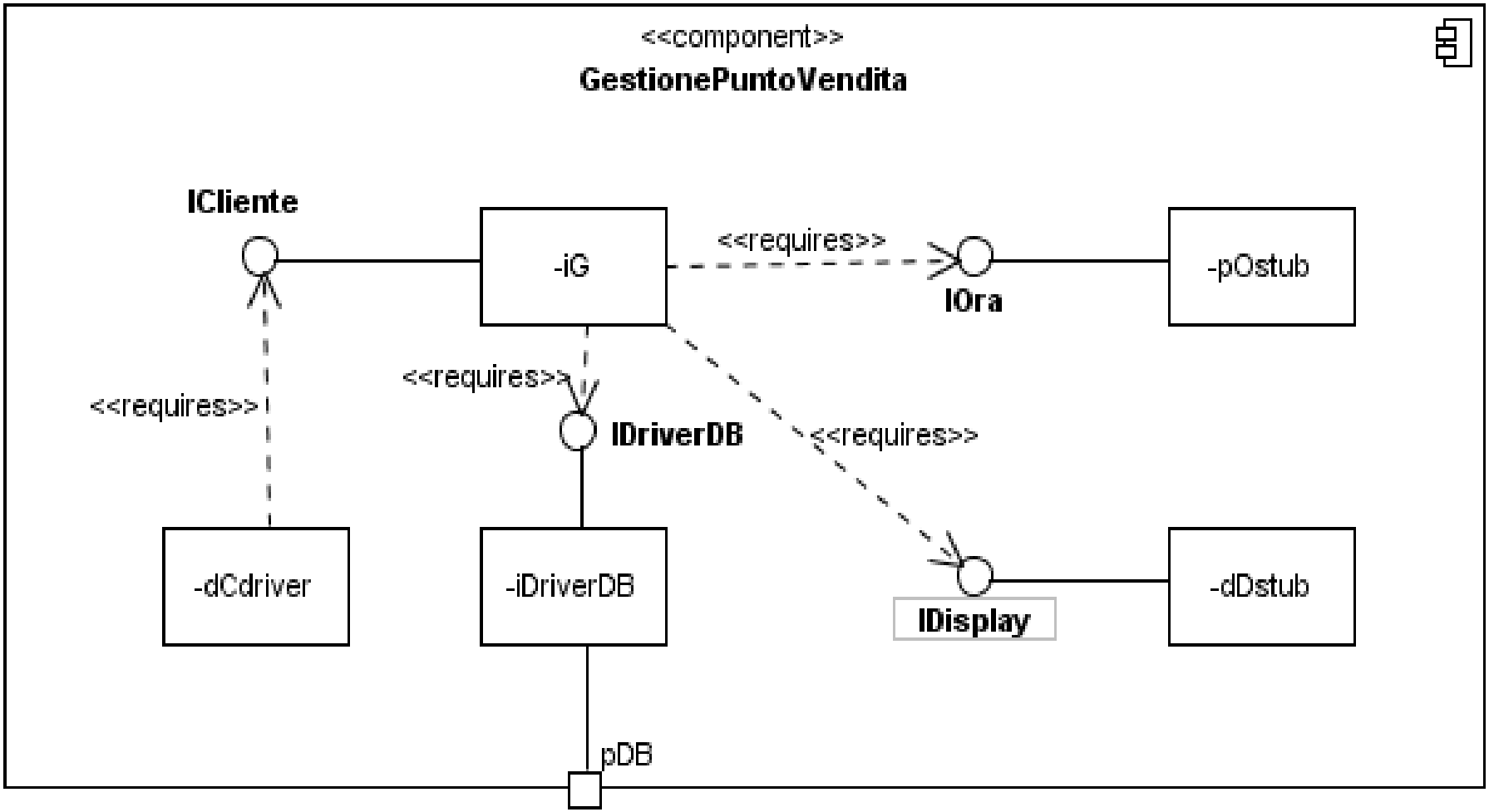
Stub e Driver

- 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



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.

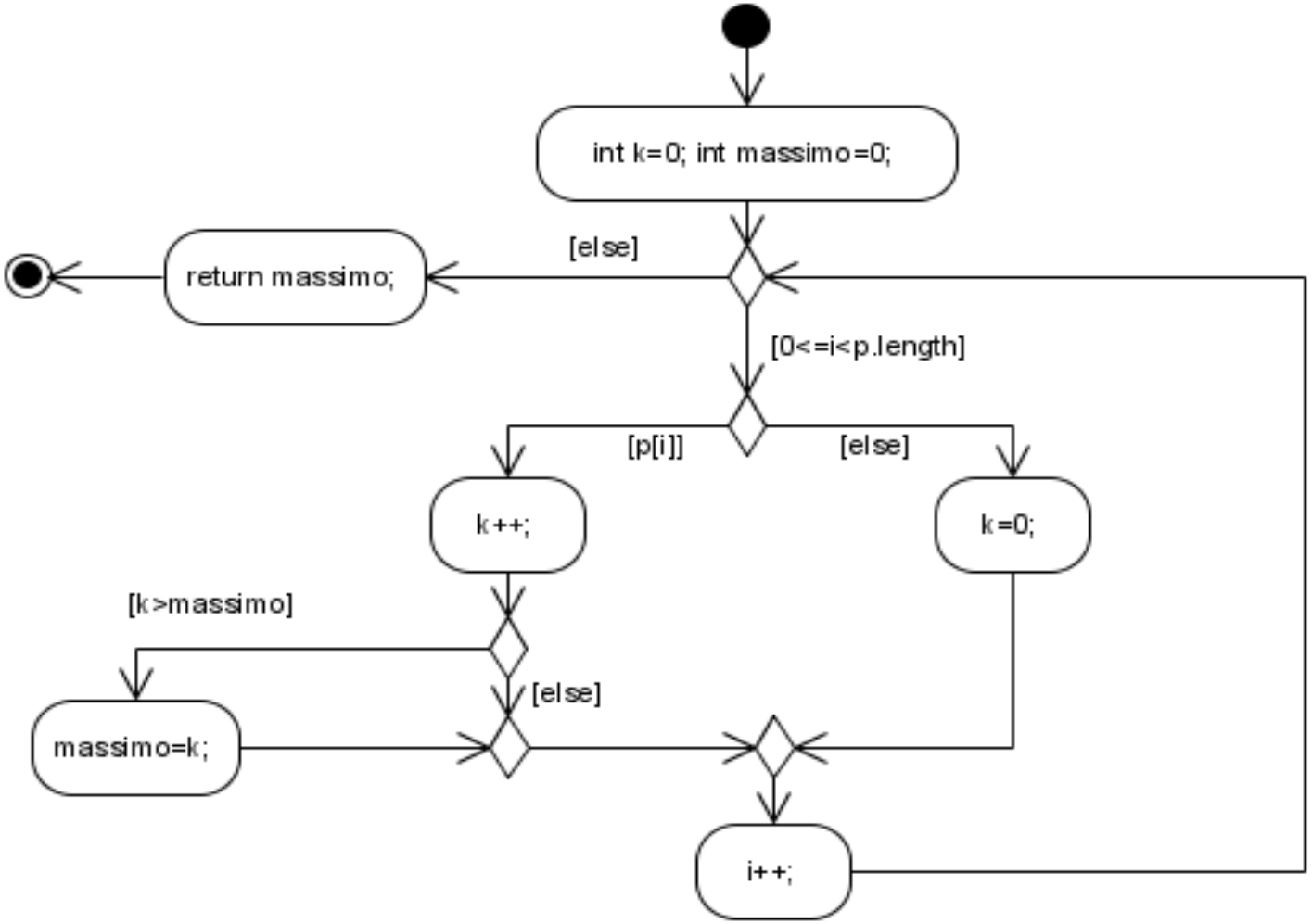
```
public int massimoPeriodo (boolean [] p) {
    int k = 0;
    int massimo = 0;
    for (int i = 0; i < p.length; i++) {
        if (p[i]) {
            k++;
            if (k > massimo) {
                massimo = k;
            }
        } else {
            k = 0;
        }
    }
    return massimo;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

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