# Politecnico di Milano

# Ingegneria del Software – a.a. 2008/09

## Appello del 25 Luglio 2008

Cognome

Nome

Matricola

Sezione (segnarne una) Baresi □, Ghezzi □, San Pietro □

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare **solo i fogli distribuiti** utilizzando il **retro** delle pagine in caso di necessità. Non separare questi fogli. Eventuali fogli di brutta, ecc. **non** verranno **in nessun caso** presi in considerazione. È possibile scrivere **in matita**.
3. È possibile consultare liberamente libri, manuali o appunti. È **proibito** l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.
6. Punteggio totale a disposizione: 25/30. La sufficienza si raggiunge con 16 punti, a cui sommare il risultato delle prove di laboratorio.

**Esercizio 1: .....**

**Esercizio 2: .....**

**Totale .....**

Esercizio 1 (punti 20)
Il gioco **Labirinto** è costituito da stanze, variamente collegate fra loro; il suo obiettivo è quello di trovare una via d'uscita, data una stanza iniziale in cui si trova il giocatore. Una classe astratta **Ambiente** fornisce l'astrazione comune alle stanze del labirinto e all'ambiente esterno in cui ci si deve portare per terminare il gioco. In altri termini, una stanza è un particolare ambiente *interno*, mentre il mondo esterno al labirinto è un particolare ambiente *esterno*. Mentre esistono più stanze, esiste solo un ambiente esterno.

1. (1 punto) Definire le intestazioni delle due classi concrete **Stanza** e **MondoEsterno**, che rappresentano gli elementi concreti di un labirinto.

2. Si consideri la classe **Labirinto**, che incapsula una struttura dati che rappresenta un insieme di ambienti e le loro connessioni. la classe fornisce un metodo osservatore **ambienti()** che restituisce gli ambienti del labirinto come **ArrayList<Ambiente>**, e un metodo osservatore **ambientiCollegati()** che restituisce in un **ArrayList<Ambiente>** l'insieme di ambienti collegati a un ambiente dato:

```
//@ ensures (*\result e' un elenco degli ambienti componenti il labirinto this *);
public /*@ pure @*/ ArrayList<Ambiente> ambienti ()

//@ ensures (*\result e' un elenco degli ambienti collegati direttamente a x*);
public /*@ pure @*/ abstract ArrayList<Ambiente> ambientiCollegati(Ambiente x);
```

2.1 (3 punti) Definire un invariante per **Labirinto** che imponga le seguenti condizioni:

a) Nessun ambiente può essere collegato a sé stesso.
b) La relazione di collegamento è simmetrica, ovverosia se l'ambiente x è collegato all'ambiente y allora y è collegato a x.
c) ci deve essere sempre uno, e uno solo, mondo esterno.

**NB: Si consiglia di utilizzare, anche nel seguito dell'esercizio, il metodo boolean contains(Object o) degli ArrayList, che restituisce *true* se, e solo se, *o* è un elemento di *this*. Non è necessario che l'invariante sia eseguibile.**

**Versione non eseguibile:**

```
/*@public invariant
        (\forall Ambiente x; ambienti().contains(x);
                !ambientiCollegati(x).contains(x) //(a)
                (\forall Ambiente y; ambienti().contains(y) && x!=y;
                        ambientiCollegati(x).contains(y) ==> ambientiCollegati(y).contains(x))) &&
        (\num_of MondoEsterno m; ; ambienti().contains(m)) ==1;
```

**Versione eseguibile:**

```
/*@public invariant
        (\forall int i; 0 <=i && i<ambienti().size();
                ambienti().get(i) != null &&
                !ambientiCollegati(ambienti().get(i)).contains(ambienti.get(i)) &&
                (\forall int j; 0<= j && j<ambienti().size() && j!= i;
                        ambientiCollegati(ambienti().get(i)).contains(ambienti.get(j)) ==>
                        ambientiCollegati(ambienti().get(j)).contains(ambienti.get(i)))) &&
        (\num_of int h; 0<=h && h<ambienti().size(); ambienti().get(h) instanceof Mondo Esterno)
==1;
```

2.2 (1 punto) Di che tipo di invariante si tratta? In quali momenti dell'esecuzione del programma deve essere vero il predicato definito dall'invariante?

**Di un invariante astratto, valido all'entrata e all'uscita dai metodi pubblici (o anche da quelli privati che non siano helper)**

2.3 (1 punto) Fornire la specifica JML del costruttore **Labirinto** vuoto, che crea un labirinto costituito solo dal mondo esterno

```
\\@ensures ambienti().size()== 1 && ambienti().get(0) instanceof MondoEsterno;
```

2.4 (2 punti) Fornire la specifica JML del metodo *public* **void addAmbiente(Ambiente a)**, che aggiunge un ambiente al labirinto. Deve trattare il caso in cui *a* sia null, deve salvaguardare l'ipotesi che non ci sia più di un mondo esterno per labirinto e inoltre deve mantenere gli ambienti pre-esistenti nel Labirinto.

```
//@requires a != null &&! (a instanceof mondo Esterno);
//@ensures (\forall Ambiente a1; ;
                    ambienti().contains(a1) <==>
                    (\old(ambienti().contains(a1)) || a == a1));
```

**In alternativa a requires, si possono lanciare eccezioni opportune.**

2.5 Fornire la specifica JML del metodo **void addConnessione(Ambiente a1, a2)** che collega a1 e a2, rispondendo alle 4 domande dettagliate che seguono. La specifica deve trattare il caso in cui a1 e a2 siano *null,* salvaguardando l'ipotesi che nessun ambiente è collegato a se stesso e che la relazione di collegamento sia simmetrica. Tutte le connessioni già esistenti tra ambienti restano inalterate.

- (3 punti) Fornire la specifica assumendo che sia irrilevante l'eventuale pre-esistenza di un collegamento tra a1 e a2. Ciò che importa è che il collegamento sussista dopo l'esecuzione del metodo.
    - **\\@requires a1!= null && a2!=null && a1 != a2 && ambienti.contains(a1) && ambienti.contains(a2);**
    - **\\@ensures**
    - **(\forall Ambiente a3; ;**
        - **(\forall Ambiente a4) ; ;**
            - **ambientiCollegati(a3).contains(a4) <==>**
            - **(\old(ambientiCollegati(a3).contains(a4)) ||**
            - **a1==a3 && a2==a4 ||**
            - **a1==a4 && a2 == a3)**

- (1 punto) Che cosa cambia se invece si assume che sia a carico del client verificare a priori la non esistenza di un collegamento tra a1 e a2 prima di invocare il metodo. La non pre-esistenza del collegamento è pertanto una pre-condizione.

  **\\@requires ! ambientiCollegati(a1).contains(a2) && …**

- (1 punto) Che cosa cambia nella specifica se invece nel caso di pre-esistenza di un collegamento tra a1 e a2 il metodo deve sollevare un'eccezione?

- **\\@signals (GiaCollegatiException e)**
  **(ambientiCollegati(a1).contains(a2) || (ambientiCollegati(a2).contains(a1);**

- (1 punto) È necessario che la specifica del metodo definisca esplicitamente la proprietà di simmetria del collegamento oppure ciò è garantito a priori dall'invariante di cui al punto 2?

  **La specifica fornita per i metodi deve rendere vero l'invariante astratto, e non viceversa.**

2.6 (2 punti) Fornire la specifica JML del metodo **boolean raggiungibile (Ambiente a1, a2)**, che fornisce un risultato vero se, e solo se, a2 è raggiungibile da a1 mediante una connessione diretta o indiretta. Deve trattare il caso in cui a1 e a2 siano null. Per definizione, se a1 e a2 coincidono (ossia a1==a2) allora a2 è raggiungibile da a1 (la raggiungibilità è una relazione riflessiva).

**//@ensures a1==a2 || (\exists Ambiente a; ambientiCollegati(a1).contains(a) && raggiungibile(a,a2)**

2.7 (1 punto) Fornire la specifica JML del metodo **boolean esisteUscita(Ambiente a)**, che fornisce un risultato vero se, e solo se, da *a* è possibile uscire dal labirinto, raggiungendo l'ambiente esterno. Deve trattare il caso in cui *a* sia *null*.

**//@ensures (\exists MondoEsterno m; ambienti().contains(m); raggiungibile(a,m));**

3. (3 punti) Si supponga che il metodo **ambientiCollegati** della classe **Labirinto** restituisca, invece dell'ArrayList, un iteratore con il quale accedere tutti gli ambienti collegati a un certo ambiente. Definire la classe Labirinto, che esporta il nuovo metodo **ambientiCollegati**. Si mostri anche l'implementazione della classe **AmbientiGen** che realizza l'iteratore. Non è necessario mostrare l'implementazione dei metodi dell'iteratore, ma solo la specifica della classe **AmbientiGen**.

4. (2 punti) Fornire un diagramma UML che descrive Labirinto, Ambiente, Stanza e MondoEsterno.