

Linguaggi di Programmazione per la Sicurezza

Esame dell'8 luglio 2011 - (Prova di laboratorio)

1 Filosofi a cena

Mille filosofi siedono ad una tavola rotonda con un piatto di spaghetti davanti, una forchetta a destra ed una forchetta a sinistra: la forchetta destra di un filosofo corrisponde alla forchetta sinistra del filosofo alla sua destra.

La vita di un filosofo consiste in periodi in cui mangia alternati a periodi in cui pensa.

Ogni filosofo ha bisogno di due forchette per mangiare; può mangiare se la forchetta alla sua sinistra e quella alla sua destra sono libere.

Se un filosofo sta pensando non mangia e se mangia non pensa.

1.1 Classe Java [pt. 2]

Scrivere una classe *DiningPhilosophers* che permetta di modellare il problema dei filosofi a cena.

All'inizio tutti i filosofi stanno pensando e le forchette sono tutte libere.

Metodo *startEating* La classe deve disporre di un metodo *startEating(int philo)* la cui implementazione viene mostrata in Figura 1.

```
public void startEating(int philo) {  
    if (!forkUsed[philo] && !forkUsed[(philo + 1)%1000]) {  
        eating[philo] = true;  
        forkUsed[philo] = true;  
        forkUsed[(philo + 1)%1000] = true;  
    }  
}
```

Figure 1: Metodo *startEating*

Il metodo *startEating* controlla se la forchetta sinistra e la forchetta destra del filosofo *philo* sono libere: nel caso le occupa e permette al filosofo di cominciare a mangiare.

Metodo *startThinking* La classe deve disporre di un metodo *startThinking(int philo)* che permetta al filosofo *philo* di cominciare a pensare. Nel caso in cui il filosofo stia già pensando lo stato della sua forchetta sinistra e destra non deve essere modificato.

Linee guida per la scrittura del codice

- Non è consentito usare campi e/o metodi statici;
- non è possibile aggiungere altri metodi rispetto a quelli descritti in precedenza.

2 Coverage

In JUnit, per il metodo *startEating*, scrivere:

- un test set che soddisfi la copertura delle decisioni, ma che non soddisfi anche la copertura delle condizioni; [pt. 2]
- un test set che soddisfi la copertura delle condizioni. [pt. 2]

Scrivere una classe JUnit per ogni test set che soddisfa un criterio di copertura.

Per ogni test case scrivere un commento nel codice che indichi cosa si vuole coprire con quel test case.

Non è necessario eseguire alcun controllo (nessuna *assert*) all'interno dei test case.

3 JML

Scrivere in JML la seguente postcondizione al costruttore:

- tutti i filosofi pensano e tutte le forchette sono libere. [pt. 1]

Scrivere in JML la seguente preconditione al metodo *startEating(int philo)*:

- l'indice *philo* è un indice valido, cioè indica uno dei mille filosofi; [pt. 0.25]

Scrivere in JML le seguenti postcondizioni al metodo *startEating(int philo)*:

- non esistono due filosofi vicini che stanno mangiando; [pt. 2.25]
- almeno una tra la forchetta sinistra e la forchetta destra del filosofo *philo* è occupata. [pt. 0.5]

Totale punti = 2 + 4 + 4 = 10

```
package esami;
```

```
public class DiningPhilosophers_20110708 {
    /*@ spec_public @*/ private boolean[] eating;
    /*@ spec_public @*/ private boolean[] forkUsed;

    //tutti i filosofi pensano e tutte le forchette sono libere
    //@ ensures (\forall int i; i >= 0 && i < 1000; !eating[i] && !forkUsed[i]);
    public DiningPhilosophers_20110708() {
        eating = new boolean[1000];
        forkUsed = new boolean[1000];
    }

    //l'indice philo e' un indice valido
    //@ requires philo >= 0 && philo < 1000;
    //non esistono due filosofi vicini che stanno mangiando
    //@ ensures !(\exists int i; i >= 0 & i < 1000; eating[i] && eating[(i + 1)%1000]);
    //almeno una tra la forchetta sinistra e la forchetta destra del filosofo philo e' occupata
    //@ ensures forkUsed[philo] || forkUsed[(philo + 1)%1000];
    public void startEating(int philo) {
        //System.out.println("!forkUsed[philo] = " + !forkUsed[philo] + "\t!forkUsed[(philo + 1)%1000] = " +
        !forkUsed[(philo + 1)%1000]);
        if(!forkUsed[philo] && !forkUsed[(philo + 1)%1000]) {
            eating[philo] = true;
            forkUsed[philo] = true;
            forkUsed[(philo + 1)%1000] = true;
        }
    }

    public void startThinking(int philo) {
        if(eating[philo]) {
            eating[philo] = false;
            forkUsed[philo] = false;
            forkUsed[(philo + 1)%1000] = false;
        }
    }

    public static void main(String[] args) {
        //viola la precondition "l'indice philo e' un indice valido"
        //DiningPhilosophers_20110708 dp = new DiningPhilosophers_20110708();
        //dp.startEating(1001);
    }
}
```

```
package esami;

import org.junit.Test;

public class DiningPhilosophers_20110708_Condizioni {

    //"!forkUsed[philo]" a true
    //"!forkUsed[(philo + 1)%1000]" a true
    @Test
    public void testCondizioni1() {
        DiningPhilosophers_20110708 t = new DiningPhilosophers_20110708();
        t.startEating(0);
    }

    //"!forkUsed[philo]" a false
    @Test
    public void testCondizioni2() {
        DiningPhilosophers_20110708 t = new DiningPhilosophers_20110708();
        t.startEating(0);
        t.startEating(1);
    }

    //"!forkUsed[(philo + 1)%1000]" a false
    @Test
    public void testCondizioni3() {
        DiningPhilosophers_20110708 t = new DiningPhilosophers_20110708();
        t.startEating(1);
        t.startEating(0);
    }
}
```

```
package esami;

import org.junit.Test;

public class DiningPhilosophers_20110708_Decisioni {

    /*!forkUsed[philo] && !forkUsed[(philo + 1)%1000]" a true
    @Test
    public void testDecisioni1() {
        DiningPhilosophers_20110708 t = new DiningPhilosophers_20110708();
        t.startEating(0);
    }

    /*!forkUsed[philo] && !forkUsed[(philo + 1)%1000]" a false
    @Test
    public void testDecisioni2() {
        DiningPhilosophers_20110708 t = new DiningPhilosophers_20110708();
        t.startEating(0);
        t.startEating(1);
    }
}
```