POLITECNICO
MILANO 1863

Esame di Ingegneria del Software

Corso 085885
Prof. Giovanni Ennio Quattrocchi

Data: 09-01-2023

Nome e Cognome: .....
Matricola: .....

Durata dell'esame: 2 ore e 30 minuti
Punteggio massimo: 33 punti. Gli studenti possono scrivere in penna o in matita per rispondere alle domande. NON è consentito consultare il materiale didattico.

Esercizio 1 – Multithreading (10 punti)

La classe `SciCommons` espone un metodo statico `isGreater` che prende in input due matrici A e B di numeri interi di grandezza `(N,M)` di tipo `List<List<Integer>>` (i.e., una lista di colonne di interi), e che produce in output una matrice C `List<List<Boolean>>` di grandezza `(N,M)` che contiene la comparazione ">" tra ciascun elemento delle matrici. Sia *x* l'elemento in posizione *i,j* della matrice A, e *y* l'elemento in posizione *i,j* della matrice B, l'elemento in posizione *i,j* della matrice C è equivalente al valore dell'espressione x > y.

Ad esempio:

```
      1  5  99          3  1  99        false  true  false
A =  -2 14  7     B =  4  7  10    C =  false  true  false
      4  9  11         14 -6  3         false  true  true
```

Per velocizzare il calcolo, il metodo `isGreater` è implementato con il multithreading. Il metodo crea un numero di thread pari al numero di colonne. Ciascun thread calcola indipendentemente dagli altri il valore di una singola di colonna della matrice C. Quando tutti i thread hanno terminato, il metodo compone la matrice C risultante e ne restituisce il valore.

Si implementi il metodo `isGreater` *creando, se necessario, un insieme di classi/interfacce a supporto di questo. Si noti che le matrici in input non devono essere modificata dal metodo.*

Esercizio 2 – Testing (6 punti)

Si consideri il seguente frammento di codice, supponendo che tutte le variabili usate siano di tipo intero o array di interi:

```
1   L = [99, 0, 4, 0, 1, 4]
2   a = 0
3   b = 1
4   if (w >= 5)
5     a -= 2
6   else
7     a += 1
8   if (y >= 1)
9     a += a*a
10  if (z >= 1)
```

w y z a b L[a]

① 4 0 0 1 0 0
② 4 0 1 3 2 0
③ 4 1 0 2 0 4
④ 4 1 1 4 2 1
⑤ 5 0 0 -2 0 \ → OUT OF INDEX
⑥ 5 0 1 0 2 99
⑦ 5 1 0 2 0 4
⑧ 5 1 1 2 2 4

F F F ①
F F T ⑥
F T F ③
F T T ⑤
T F F ⑤
T F T ⑥
T T F ⑦
T T T ⑧

viene sempre rilevato dalla cop. dei cammini

```
11      a += ++b    b=2
12    else
13      a -= b--    b=0
14    return L[a]
```

a) *(4 punti) Si definisca un insieme* ***minimo*** *di test utilizzando il criterio di* ***copertura dei cammini****.*
b) *(2 punti) Si descriva quali errori presenta il frammento di codice e si spieghi se tali errori vengono sempre rilevati da un insieme minimo di* ***copertura dei cammini****.*

Esercizio 3 - UML (11 punti)

Si vuole implementare un gioco di strategia. Il gioco è costituto da diversi giocatori (massimo 4) rappresentati da un colore ed una relativa pedina, da un tabellone lineare (dalla casella 1 alla 100), e da un mazzo di carte. Vince il primo giocatore che raggiunge la casella 100. Ad ogni turno ciascun giocatore deve lanciare un dado. Il giocatore può a questo punto effettuare diverse azioni (una sola per turno): i) spostare la sua pedina di un numero di posizioni pari a quello sul dado, ii) pescare una carta (solo se il dado è superiore o uguale a 5), iii) giocare una carta, iv) passare il turno. Si considerino tali azioni personalizzabili, ad esempio da una futura espansione del gioco. Le caselle possono essere di diverso tipo: quelle normali oppure possono avere un effetto (ad esempio, far pescare una carta al giocatore, farlo retrocedere o avanzare di ulteriori caselle). Le carte hanno un intervallo di attivazione basato sulla posizione della pedina (ad esempio dalla casella 10 alla 20), un valore, ed un operatore (ad esempio: +, -, *). Il giocatore può giocare una carta solo se la sua posizione nel tabellone è incluso nell'intervallo, e la sua posizione varierà a seconda del valore della carta e dell'operatore (ad esempio, se l'operatore è + e il valore è 3, la pedina avanzerà di 3 caselle).

Si definisca un Class Diagram, ***utilizzando opportuni design pattern,*** *che descriva le entità rilevanti del dominio applicativo, indicando metodi e attributi significativi e le associazioni tra entità. Si assegni un nome a ciascuna associazione e si forniscano indicazioni sui vincoli di cardinalità. Si commentino le scelte effettuate quando necessario.*

Esercizio 4 – Generics (6 punti)

Si consideri il metodo `isGreater` *dell'Esercizio 1 e si crei una versione senza l'uso del multithreading ma che supporti matrici di qualunque tipo T e non solo di* `Integer`*. Si creino eventuali classi/interfacce aggiuntive, se necessario. È inoltre consentito l'uso dei componenti messi a disposizione dal linguaggio Java.*

Si noti che nella signature di un metodo è possibile limitare il type parameter in modo che questo implementi una specifica interfaccia o estenda una determinta classe attraverso la sintassi <A extends B> dove A è il type parameter e B una classe o interfaccia esistente. Ad esempio:

```
public MyInterface {
 ….
}

public <T extends MyInterface> void example(T a){
     …
}
```

Il metodo example prenderà in input un oggetto di qualunque tipo a patto che implementi l'interfaccia MyInterface.