

Prova scritta di Programmazione II

FAC-SIMILE

LEGGERE CON ATTENZIONE

- Il tempo a disposizione per lo svolgimento della prova è di **2 ore**.
- Non è consentita la consultazione di appunti, dispense, libri, ecc.
- Non è consentito l'uso di dispositivi elettronici (laptop, tablet, smartphone, e-reader, lettori MP3, ecc.).
- Al termine della prova, **consegnare il testo del compito** (il quale sarà pubblicato successivamente) e tutti i fogli protocollo contenenti esercizi da correggere.
- Ricordarsi di compilare la sezione **DATI DELLO STUDENTE** qui sotto e di **scrivere nome, cognome e matricola** su ogni foglio protocollo consegnato.

DATI DELLO STUDENTE

Nome

Cognome

Matricola

Corso (A o B)

SEZIONE RISERVATA AL DOCENTE

Esercizio 1

Esercizio 2

Esercizio 3

Esercizio 4

Esercizio 1 (8 punti) Date le classi (incomplete)

```
abstract class Tree<T> {
    public abstract Tree<T> detach(T x);
}

class Leaf<T> extends Tree<T> {
    public Tree<T> detach(T x) {
        // COMPLETARE
    }
}

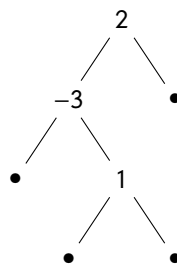
class Branch<T> extends Tree<T> {
    private T elem;
    private Tree<T> left;
    private Tree<T> right;

    public Branch(T elem, Tree<T> left, Tree<T> right) {
        this.elem = elem;
        this.left = left;
        this.right = right;
    }

    public Tree<T> detach(T x) {
        // COMPLETARE
    }
}
```

fornire le implementazioni del metodo detach in Leaf e Branch in modo tale che t.detach(x) restituisca una versione modificata di t in cui ogni sottoalbero avente radice x è stato sostituito con l'albero vuoto. Se x non è presente, il metodo deve restituire l'albero invariato.

Ad esempio, se t è l'albero



dove • indica una istanza di Leaf, allora t.detach(-3) deve restituire l'albero che contiene il solo elemento 2. Realizzare il metodo detach in modo da minimizzare il numero di nuovi oggetti creati. Non è consentito aggiungere metodi, usare cast o metodi della libreria standard di Java. Si può assumere che l'albero t non contenga elementi null.

Esercizio 2 (8 punti) Date le classi e interfacce

```
interface I {  
    public void m1(J obj);  
}  
interface J {  
    public void m2();  
}  
abstract class C implements I {  
    public abstract void m1(J obj);  
}  
class D extends C implements J {  
    public void m1(J obj) {  
        if (this != obj) obj.m2();  
        System.out.println("D.m1");  
    }  
    public void m2() {  
        System.out.println("D.m2");  
        m1(this);  
    }  
}
```

rispondere alle seguenti domande:

1. Se si eliminasse il metodo `m1` dalla classe `C`, il codice sarebbe comunque corretto? Perché?
2. Il seguente codice è corretto? Se no, spiegare perché. Se sì, determinare cosa stampa.

```
I obj = new D();  
((D) obj).m2();
```

3. Il seguente codice è corretto? Se no, spiegare perché. Se sì, determinare cosa stampa.

```
J obj = new D();  
C x = (C) obj;  
x.m1(new D());
```

4. Il seguente codice è corretto? Se no, spiegare perché. Se sì, determinare cosa stampa.

```
C obj = new D();  
obj.m1(obj);
```

Esercizio 3 (6 punti) Dato il codice

```
class Node<T> {  
    public T elem;  
    public Node<T> next;  
  
    public Node(T elem, Node<T> next) {  
        this.elem = elem;  
        this.next = next;  
    }  
}  
  
public static <T extends Comparable<T>> void metodo(Node<T> p, T x) {  
    while (x.compareTo(p.elem) < 0)  
        p = p.next;  
    p.next = null;  
}
```

1. Determinare sotto quali condizioni il metodo viene eseguito correttamente (cioè senza lanciare alcuna eccezione) e scrivere una corrispondente **asserzione** da aggiungere come condizione per il metodo. Nello scrivere l'asserzione è possibile fare uso di eventuali metodi statici ausiliari che **vanno comunque definiti** anche se visti a lezione.
2. Descrivere in modo conciso e chiaro, in **non più di 2 righe di testo**, l'effetto del metodo.

Esercizio 4 (8 punti) Siano date le classi

```
abstract class List {
    public abstract List reverse(List p);
}

class Nil extends List {
    public List reverse(List p) {
        // CHECK POINT 2
        return p;
    }
}

class Cons extends List {
    private int elem;
    private List next;

    public Cons(int elem, List next) {
        this.elem = elem;
        this.next = next;
    }

    public List reverse(List p) {
        List n = next;
        next = p;
        return n.reverse(this);
    }
}

class TestHeap {
    public static void main(String[] args) {
        List l = new Cons(m1, new Cons(m2, new Nil()));
        // CHECK POINT 1
        l = l.reverse(new Nil());
    }
}
```

dove m_1 ed m_2 sono le ultime 2 cifre del proprio numero di matricola. Si disegnino stack e heap al raggiungimento di ciascuno dei due check point, per un totale di 4 disegni.