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 quali laptop, tablet, smartphone, e-reader, ecc.
- Al termine della prova, **consegnare** il testo del compito e tutti i fogli protocollo contenenti esercizi da correggere.
- Ricordarsi 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) *La differenza di due liste p e q, che indichiamo con la notazione $p - q$, è la lista che contiene tutti gli elementi di p che non compaiono in q. Ad esempio:*

- $[1, 2, 3, 4, 5, 6] - [2, 4, 6] = [1, 3, 5]$
- $[1, 2, 3, 4, 5, 6] - [] = [1, 2, 3, 4, 5, 6]$
- $[1, 1, 1] - [1] = []$
- $[2, 4, 6] - [1, 2, 3, 4, 5, 6] = []$

Date le classi (incomplete)

```
class Node {
    public int elem;
    public Node next;

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

public class Main {
    public static Node diff(Node p, Node q) {
        // COMPLETARE
    }
}
```

implementare il metodo statico ricorsivo `diff` *in modo tale che* `diff(`p`, `q`)` *ritorni $p - q$ assumendo che p e q siano* ***LISTE ORDINATE****.*

Nella risoluzione dell'esercizio tenere conto dei seguenti vincoli: `diff` *non deve mai lanciare eccezioni né modificare in alcun modo le liste su cui opera;* ***non è consentito usare cicli, cast o metodi non mostrati nel codice qui sopra****.*

Prestare attenzione all'eventualità che le due liste contengano elementi duplicati.

Esercizio 2 (8 punti) *Date le classi e interfacce*

```
interface A {
    public void m1();
}

interface B {
    public void m1();
    public void m2(A obj);
}

class C implements A, B {
    public void m1()
    { System.out.println("C.m1"); }

    public void m2()
    {
        System.out.println("B.m2");
        this.m1();
    }

    public void m2(A obj)
    {
        System.out.println("B.m3");
        this.m2();
    }
}
```

rispondere alle seguenti domande:

1. *Se si eliminasse il metodo* `m1` *dall'interfaccia* `B`*, il codice sarebbe comunque corretto? Perché?*
2. *Il seguente codice è corretto? Se no, spiegare perché. Se sì, determinare cosa stampa.*

```
B obj1 = new C();
obj1.m2(obj1);
```

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

```
B obj1 = new C();
obj1.m2(new C());
```

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

```
B obj = new C();
obj.m2();
```

1) se si eliminasse il metodo m1() da B, il codice sarebbe corretto perché il metodo m1() è previsto anche nell'interfaccia A e la classe C è sottoclasse sia di A sia di B

2) non è corretto perché dà errore in runtime: infatti il metodo m2() chiede un parametro di tipo A, ma visto che obj1 ha tipo vero C, in runtime provoca questo errore

3) non è corretto perché dà errore in runtime: infatti il metodo m2() chiede un parametro di tipo A, ma noi stiamo passando un oggetto di tipo C e questo provoca l'errore

4) il codice è corretto perché viene considerato, durante la compilazione, il metodo m2() senza parametri , il quale è previsto nella classe C. Stampa B.m2 e C.m1

Esercizio 3 (6 punti) *Sia dato il metodo*

```
public static float metodo(int[] a) {
    float ris = 0;
    for (int i = 0; i < a.length; i++)
        ris += 1 / (float) a[i];
    return ris / a.length;
}
```

1. *Descrivere in modo conciso e chiaro, in **non più di 2 righe di testo**, l'effetto del metodo.*
2. *Determinare la condizione **più debole** che garantisce l'esecuzione del metodo senza eccezioni e scrivere una corrispondente **asserzione** da aggiungere come precondizione 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.*

1) Il metodo calcola la somma dei reciproci dei singoli valori dell'array di interi a e poi fa la divisione rispetto al numero di elementi dell'array a.

2) La condizione piu debole è che nell'array a, tutti gli elementi devono essere diversi da 0 e che l'array non sia vuoto o null

```
assert a.length > 0 && a != null && isThereAZero(a): "non si può calcolare"

public static boolean isThereAZero(int[] a){
    boolean ris = true;
    for(int i = 0; i < a.length && ris; i++){
        ris = ris && a[i] != 0;
    }
    return ris;
}
```

Esercizio 4 (8 punti) *Siano date le classi*

```
abstract class List {
    public abstract List insert(int n);
    public abstract List sort();
}

class Nil extends List {
    public List insert(int x)
    { return new Cons(x, this); }

    public List sort() {
        return this;
    }
}

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

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

    public List insert(int x) {
        // CHECK POINT 2
        if (x < elem) return new Cons(x, this);
        else if (x == elem) return this;
        else return new Cons(elem, next.insert(x));
    }

    public List sort() {
        List l = next.sort();
        return l.insert(elem);
    }
}

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

dove m_1 ed m_2 sono le prime 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.