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 List {
    public abstract double eval(double x);
}

class Nil extends List {
    public double eval(double x) {
        // IMPLEMENTARE
    }
}

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

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

    public double eval(double x) {
        // IMPLEMENTARE
    }
}
```

fornire le implementazioni del metodo `eval` *in* `Nil` *e* `Cons` *in modo tale che, se* `l` *è la lista che contiene i valori* $[a_0, \ldots, a_n]$*, allora* `l.eval(`x`)` *calcola*

$$\sum_{i=0}^{n} a_i x^i$$

Ad esempio, se `l` *è la lista* $[1, 2, 3]$ *allora* `l.eval(5)` *deve calcolare* $1 \cdot 5^0 + 2 \cdot 5^1 + 3 \cdot 5^2 = 1 + 10 + 75 = 86$*. Non è consentito aggiungere metodi, usare cast o metodi della libreria standard di Java.*

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

```
interface I {
    public void m1(J obj);
}
interface J {
    public void m2(I obj);
}
interface K {
    public void m3();
}
class C implements I, J {
    public void m1(J obj) {
        obj.m2(this);
        System.out.println("C.m1");
    }
    public void m2(I obj) {
        obj.m1(this);
        System.out.println("C.m2");
    }
}
class D implements J, K {
    public void m1(J obj)
    { System.out.println("D.m1"); }
    public void m2(I obj)
    { System.out.println("D.m2"); }
    public void m3() {
        m1(this);
        System.out.println("D.m3");
    }
}
```

rispondere alle seguenti domande:

1. *Se si eliminasse il metodo* `m1` *dalla classe D, 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();
obj.m1(new C());
```

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

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

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

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

1) sarebbe ancora corretto poiché la classe D non estende l'interfaccia I che contiene il metodo m1() perciò non è obbligata ad avere quel metodo.
2) Non è corretto poiché la classe D non è sottoclasse dell'interfaccia I
3) Il codice è corretto poiché obj ha la classe C è sottoclasse di I e obj ha tipo apparente I, in I si trova il metodo m1(J obj). All'interno della chiamata abbiamo un oggetto di tipo D, sottoclasse di J,

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

```
public static int metodo(int[] a, int[] b) {
    int s = 0;
    for (int i = 0; i < b.length; i++)
        s += a[b[i]];
    return s;
}
```

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.*

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

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

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

    public List onto(List l)
    {
        // CHECK POINT 2
        return l;
    }
}

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) {
        if (x < elem) return new Cons(x, this);
        else if (x == elem) return this;
        else return new Cons(elem, next.insert(x));
    }

    public List onto(List l)
    { return next.onto(new Cons(elem, l)); }
}

class TestHeap {
    public static void main(String[] args) {
        List l = new Nil();
        l = l.insert(m1); // m1 = prima cifra del numero di matricola
        l = l.insert(m2); // m2 = seconda cifra del numero di matricola
        // CHECK POINT 1
        l = l.onto(new Nil());
    }
}
```

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.