COGNOME: **NOME:** **MATRICOLA:**

Esame di Programmazione per BioInformatica, 17-giu-2015

Esercizio 1

Si consideri la seguente classe per le liste:

```
public class List {
    private int head;
    private List tail;
    public List(int head, List tail) {
         this.head = head;
         this.tail = tail;
    }
}
```

Si aggiunga un metodo di istanza `public List getOrdered()` **ricorsivo**, che restituisca la lista ordinata in modo **decrescente** a partire dalla lista `this`. La lista `this` non deve comunque essere modificata. Ad esempio, se `this` è `[3;2;4;8;2]`, allora `getOrdered()` deve restituire la lista `[8;4;3;2;2]`.

La soluzione deve essere *ricorsiva*: è **vietato** usare i costrutti **while, for o do while**. Il metodo `getOrdered()` può invocare altri metodi eventualmente aggiunti alla classe (l'utilizzo dei costrutti while, for o do while comporta la completa nullitá dell'esercizio).

Esercizio 2

Si consideri la seguente classe che realizza una stack:

```
public class Stack {
    private int pos;
    private int s[];
    public Stack(int n) {
         this.pos  = 0;
         this.s    = new int[n];
    }
    public Stack() {
         this.pos  = 0;
         this.s    = new int[10];
    }
}
```

Si aggiungano i metodi `public push(int i)` e `public int pop()` che rispettivamente inserisce ed estrae un elemento dalla testa dello stack.
Si gestiscano inoltre le situazioni di stack pieno o vuoto. (Es. Estrazione dallo stack vuoto) Qualora si ritenga si aggiungano ulteriori metodi alla classe e si motivino le scelte fatte.

Esercizio 3

Si considerino le seguenti classi (ogni classe è in un file separato)

```
public class X {
    int a = 0;
    public X() {}
    public boolean isX(Object v) { return (v instanceof X); }
    public void f(int a) {
        System.out.println("sono f(int a) di X" + " " + a + " " + this.a);
    }
    public void f(long a){
        System.out.println("sono f(long a) di X" + " " + a + " " + this.a);
    }
    public void g(long a){
        System.out.println("sono g(long a) di X" + " " + a + " " + this.a);
    }
}
```

(continua)

```
public class Y extends X {
    int a = 1;
    public void g(int a) {
        System.out.println("sono g(int a) di Y" + " " + a + " " + this.a);
    }
    public void f(long a) {
        System.out.println("sono f(long a) di Y" + " " + a + " " + this.a);
    }
}

public class XYmain{
    public static void main(String[] args) {
        X x = new X();
        Y y = new Y();
        X z = y;
        System.out.println(x.isX(z) + " " + y.isX(x));
        x.f(999);
        x.f(888L);
        y.f(777);
        y.f(666L);
        z.f(555);
        z.f(444L);
        z.g(-1);
        y.g(-2);
    }
}
```

1. cosa stampa il metodo `main()`?
2. per ogni invocazione in `main()` di uno dei metodi delle classi `X` e `Y`, si indichi:
   (a) la segnatura individuata in fase di compilazione;
   (b) l'implementazione del metodo eseguita in fase di esecuzione (indicate la classe ove tale implementazione è definita e dite se tale implementazione è stata ereditata).

Aiuto 1 Ricordo che il comando `instanceof` restituisce true/false a seconda che un oggetto si un'istanza di una certa classe.
Esempio:

```
String s = "Ciao";
Integer i;

s instanceof String => true
i instanceof String => false
```

Soluzione 1

```
public class List {
    private int head;
    private List tail;

    public List(int head, List tail) {
        this.head = head;
        this.tail = tail;
    }
```

```
/* E' necessario scrivere un metodo ricorsivo che inserisca un elemento in modo
   Ordinato nella lista lincata */
    public void insertOrd(int val){
        if (this.head < val){ // il valore da inserire viene dopo del corrente
                              // Cambiando il verso del confronto
                              // Cambia l'ordinamento crescente
                              // o decrescente
            this.tail = new List(this.head, this.tail);
            this.head = val;
        }
        else if (this.tail == null) this.tail = new List(val, null);
        else tail.insertOrd(val);
    }


/* La soluzione vera e propria  che si appoggia sul metodo precedente */
    private void getOrdered(List l){
        if (this.tail != null) {
            l.insertOrd(this.tail.head);
            this.tail.getOrdered(l);
         }
    }

    public List getOrdered() {
        List l = new List(this.head, null); // Creo nuova lista con primo elemento
        getOrdered(l);
        return l;
    }
```

```
/**** Versione toString() ricorsiva ******  QUESTA PARTE NON E' RICHIESTA
        INSERITA SOLO PER COMPLETEZZA
    private String convert(){
        if ( tail == null ) return "" + head;
        else return "" + head + ";" + tail.convert();
    }

    public String toString(){
       return "[" + convert() + "]";
    }
****************************************/

/**** Versione iterativa ****************/

 public String toString(){
       String s = "[" + this.head;
       List l = this.tail;
```

```
        while (l != null){
            s = s + ";" + l.head;
            l = l.tail;
         }
         s = s + "]";
         return s;
     }

      public static void main(String[] args) {
         List l = new List(3,new List(2,new List(4,
                     new List(5,new List(6,new List(8,
                     new List(2,new List(3,new List(7,
                     new List(8,new List(9,null)))))))))));

         List n = l.getOrdered();
         System.out.println(l);
         System.out.println(n);
     }
}
```

```
OUTPUT: [3;2;4;5;6;8;2;3;7;8;9] [9;8;8;7;6;5;4;3;3;2;2]
```

Soluzione 2

```
public class Stack {
  private int pos;
  private int s[];

  public Stack(int n) {
    this.pos = 0;
    this.s = new int[n];
  }

  public Stack() {
    this.pos = 0;
    this.s = new int[10];
  }
}
```

```
   public void push(int i){
     if (pos >= s.length){
                int tmp[] = new int[s.length * 2];
                int j;
                for (j = 0; j < s.length; j++)
                  tmp[j] = s[j];
                s = tmp; // sovrascive la varibile
         }
         s[pos] = i;
         pos = pos + 1;
   }

   public int pop() throws EmptyStackException{
      if (pos > 0) {
        pos = pos - 1;
        return s[pos];
      }
      else throw new EmptyStackException("Stack Empty");

   }
```

```
public class EmptyStackException extends RuntimeException
{

    public EmptyStackException(String s)
    {
        // initialise instance variables
       super(s);
    }
}
```