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

Compito di Programmazione, 22 marzo 2007 – INFORMATICI E MULTIMEDIALI

Esercizio 1 Si consideri la seguente classe che specifica liste di interi:

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

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

Si aggiunga a `List` un metodo ricorsivo (o che chiama un metodo ricorsivo) che restituisce la lista dei massimi locali contenuti in `this`, nell'ordine in cui compaiono in `this`. Un massimo locale di una lista è un elemento che è strettamente più grande di quello che lo precede e di quello che lo segue. Il primo e l'ultimo elemento di una lista non sono mai massimi locali perchè non hanno un elemento che precede o segue, rispettivamente.

(continua)

Esercizio 2 Si progetti una classe di nome `GrandiIntPositivi`, la quale istanzia oggetti che modellano numeri interi positivi di 20 cifre. La classe mantiene cifre comprese tra 0 e 9 in un campo array privato `int[] cifre` di dimensione costante `MAX` posta a 20 e fornisce:

1. un costruttore che, invocato senza parametri, costruisce un oggetto che rappresenta lo zero, ovvero inizializza l'array ponendo tutti gli elementi a zero;

2. un costruttore che, ricevuto un parametro di tipo `String`, costruisce un oggetto che rappresenta il numero contenuto nella stringa. Per semplicità si assuma che la stringa contenga solo cifre decimali. Inoltre, lo stesso costruttore sia eventualmente in grado di scartare la coda di stringhe lunghe più di 20 cifre;

3. un metodo `GrandiIntPositivi piu(GrandiIntPositivi g)` che restituisce, in forma di un nuovo oggetto, la somma dell'oggetto che esegue il metodo con quello passato come parametro. Se la stessa somma supera le 20 cifre allora restituisce l'oggetto che rappresenta il più grande intero positivo di 20 cifre;

4. un metodo `String toString()` che restituisce in forma di stringa lunga 20 caratteri il numero rappresentato dall'oggetto che utilizza il metodo;

5. un metodo `boolean equals(GrandiIntPositivi g)` che verifica l'uguaglianza tra due oggetti della classe in base al numero che essi rappresentano.

La classe inoltre contenga un metodo `main()` il quale, costruiti due oggetti `GrandiIntPositivi`, stampa la somma dei due oggetti e ne verifica l'uguaglianza.

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

Compito di Programmazione, 21 giugno 2007 – INFORMATICI E MULTIMEDIALI

Esercizio 1 Si scriva una classe `CercaCoppia` che fornisce:

- un costruttore `CercaCoppia(int[] arr)`

- un metodo `boolean contiene(int somma)` che determina se per l'array passato al momento della creazione esistono due indici `i,j` non necessariamente distinti tali che `arr[i] + arr[j] == somma`. Questo metodo deve essere definito ricorsivamente o usare un metodo ausiliario il quale, verificata iterativamente la possibile uguaglianza su tutti gli indici `j` per un fissato valore `i`, ricorsivamente invoca la stessa verifica per un altro valore di `i` nel caso in cui l'uguaglianza non sia verificata.

Se tutto e' definito correttamente, il seguente metodo main:

```
public static void main(String[] args) {
    ConsoleOutputManager schermo = new ConsoleOutputManager();

    int[] arr = {4,6,-2,5,11,7};

    CercaCoppia cc = new CercaCoppia(arr);

    schermo.println(cc.contiene(1));
    schermo.println(cc.contiene(2));
    schermo.println(cc.contiene(8));
    schermo.println(cc.contiene(11));
    schermo.println(cc.contiene(14));
    schermo.println(cc.contiene(15));
    schermo.println(cc.contiene(20));
}
```

deve stampare:

```
false
true
true
true
true
true
false
```

(continua)

Esercizio 2 Si progetti una classe di nome `GrandiIntPositivi`, la quale istanzia oggetti che modellano numeri interi positivi di 20 cifre. La classe mantiene cifre comprese tra 0 e 9 in un campo array privato `int[] cifre` di dimensione costante `MAX` posta a 20 e fornisce:

1. un costruttore che, invocato senza parametri, costruisce un oggetto che rappresenta lo zero, ovvero inizializza l'array ponendo tutti gli elementi a zero;

2. un costruttore che, ricevuto un parametro di tipo `String`, costruisce un oggetto che rappresenta il numero contenuto nella stringa. Per semplicità si assuma che la stringa contenga solo cifre decimali. Inoltre, lo stesso costruttore sia eventualmente in grado di scartare la coda di stringhe lunghe più di 20 cifre;

3. un metodo `GrandiIntPositivi piu(GrandiIntPositivi g)` che restituisce, in forma di un nuovo oggetto, la somma dell'oggetto che esegue il metodo con quello passato come parametro. Se la stessa somma supera le 20 cifre allora restituisce l'oggetto che rappresenta il più grande intero positivo di 20 cifre;

4. un metodo `String toString()` che restituisce in forma di stringa lunga 20 caratteri il numero rappresentato dall'oggetto che utilizza il metodo;

5. un metodo `boolean equals(GrandiIntPositivi g)` che verifica l'uguaglianza tra due oggetti della classe in base al numero che essi rappresentano.

La classe inoltre contenga un metodo `main()` il quale, costruiti due oggetti `GrandiIntPositivi`, stampa la somma dei due oggetti e ne verifica l'uguaglianza.

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

Compito di Programmazione, 26 luglio 2007 – INFORMATICI

Esercizio 1 Si progetti una classe di nome `CodiceIntero`, i cui oggetti codificano numeri interi. Ogni oggetto della classe rappresenta un codice intero le cui cifre sono ordinate nel verso opposto a quello del valore numerico da cui l'oggetto è stato creato. Se, ad esempio, il numero in questione è 74235 allora il corrispondente oggetto `CodiceIntero` rappresenta il codice 53247.

Dopo avere previsto nella classe le necessarie variabili d'istanza, si progettino i seguenti metodi:

- (costruttore/codificatore) `public CodiceIntero (int numero)`
- (decodificatore) `public int decodifica()`
- (stampa del codice) `public String toString()`

Infine, si dia un metodo `main` che a partire da un numero intero istanzia un oggetto `CodiceIntero` e, successivamente, stampa il valore del relativo codice. Infine stampa il valore del valore decodificato dal codice stesso.

Nota bene: soluzioni che si basino su preesistenti metodi di libreria che effettuano la manipolazione di stringhe NON verranno accettate. La soluzione si può ottenere utilizzando semplici operazioni su interi.

(continua)

Esercizio 2 Si completi la seguente classe per le liste:

```
import prog.io.ConsoleOutputManager;

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

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

    public String toString() {
        return "[" + toStringAux() + "]";
    }

    private String toStringAux() {
        if (tail == null) return "" + head;
        else return head + "," + tail.toStringAux();
    }

    public List compact() {
    }

    public List decompact() {
    }

    public static void main(String[] args) {
        ConsoleOutputManager schermo = new ConsoleOutputManager();

        List l =
            new List
            (5,new List
             (5,new List
              (5,new List
               (6,new List
                (4,new List
                 (4,new List
                  (3,new List
                   (3,new List
                    (3,new List
                     (3,null)))))))))));

        schermo.println("La lista era: " + l);
        schermo.println("Compressa diventa: " + l.compact());
        schermo.println("Decompressa diventa: " + l.compact().decompact());
    }
}
```

in modo che `compact()` restituisca la compressione di una lista di numeri non negativi e `decompact()` la sua decompressione. Per compressione si intende che numeri ripetuti consecutivamente almeno due volte vengono compressi indicano quante volte sono ripetuti tramite un numero negativo. Per esempio, il `main()` precedente deve stampare:

```
La lista era: [5,5,5,6,4,4,3,3,3,3]
Compressa diventa: [-3,5,6,-2,4,-4,3]
Decompressa diventa: [5,5,5,6,4,4,3,3,3,3]
```

dove `-3,5` va interpretato come “3 volte 5”, `-2,4` come “2 volte 4” e `-4,3` come “4 volte 3”.

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

Compito di Programmazione, 6 setttembre 2007 – INFORMATICI

Esercizio 1 Si consideri la seguente classe Java, che implementa un'"arena", cioè un array n x m di `Robot`:

```
public class Arena {

    // gli elementi dell'arena
    private Robot[][] posizione;

    // costruisce un'arena n x m, inizialmente vuota
    public Arena(int n, int m) {
        posizione = new Robot[n][m];
    }

    // mette un robot dentro questa arena alle coordinate indicate
    public void metti(Robot robot, int x, int y) {
        if (x < 0 || x >= posizione.length ||
            y < 0 || y >= posizione[x].length) return;

        posizione[x][y] = robot;
    }

    // restituisce il robot contenuto in questa arena nelle coordinate indicate
    public Robot contenuto(int x, int y) {
        if (x < 0 || x >= posizione.length ||
            y < 0 || y >= posizione[x].length) return null;

        return posizione[x][y];
    }

    // main di prova
    public static void main(String[] args) {
        Arena arena = new Arena(10,15);

        new Robot(arena,5,7);

        // questo si sposta in 6,6
        new Robot(arena,8,2).sposta(6,6);

        new RobotGentile(arena,9,0);

        // questo rimane in 8,7
        new RobotGentile(arena,8,7).sposta(6,6);
    }
}
```

Si scriva una classe `Robot.java` completando il seguente schema:

```
public class Robot {

    ....

    // crea un robot e lo posiziona nell'arena indicata e nelle
    // coordinate indicate; se li' c'era un altro robot, lo sovrascrive
    public Robot(Arena arena, int x, int y) {
        .....
    }
```

```
    // restituisce l'arena in cui e' stato creato questo robot
    protected Arena getArena() {
        .....
    }

    // sposta questo robot nelle nuove coordinate indicate;
    // se li' c'era un altro robot, lo sovrascrive
    public void sposta(int nuovaX, int nuovaY) {
        .....
    }
}
```

Si scriva quindi una sua sottoclasse `RobotGentile.java` che si comporta come `Robot.java` tranne per il fatto che non si sposta in una posizione già occupata da un altro robot.

Tutti i campi delle classi devono essere dichiarati `private`.

Esercizio 2 Si progetti una classe di nome `Numeri`, i cui oggetti specificano un array `a` di stringhe di dimensione 100. Ciascun elemento dell'array contiene una stringa binaria il cui simbolo iniziale è `1`, mentre tutti i restanti simboli sono `0`.

Dopo avere previsto nella classe le necessarie variabili d'istanza, si progettino i seguenti metodi:

- `public Numeri (String numero)`, il quale attraverso l'uso di una procedura ricorsiva (che eventualmente si appoggi su un metodo ausiliario) assegna agli elementi dell'array stringhe ottenute estraendo via via dal parametro `numero` sottostringhe secondo il formato previsto. Se, ad esempio, `numero = ¨10011000¨`, allora si avrà `a[0]=¨100¨`, `a[1]=¨1¨`, `a[2]=¨1000¨`, `a[3]=null`, ..., `a[99]=null`.
- `public String toString()`, il quale restituisce una stringa formata concatenando gli elementi dell'array (esclusi i riferimenti `null`) attraverso la sequenza di escape *newline*. Nell'esempio precedente, `toString()` restituirà

  ```
  100
  1
  1000
  ```

Infine, si dia un metodo `main` che a partire da una stringa binaria obbligatoriamente iniziata dal simbolo `1` istanzia un oggetto `Numeri` e, successivamente, stampa l'oggetto.

Nota bene: soluzioni che si basino su algoritmi di tipo iterativo per la creazione dell'oggetto NON verranno accettate.

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

Compito di Programmazione, 27 setttembre 2007 –

Esercizio 1 Si progetti una classe di nome `Treno`, i cui oggetti modellano un treno formato da una locomotiva e n vagoni, con $n \geq 0$, ciascuno contenente p_i passeggeri con $1 \leq i \leq n$.

Dopo avere previsto nella classe le necessarie variabili d'istanza, si progettino i seguenti metodi:

- `public Treno()`, il quale costruisce un treno formato dalla sola locomotiva;
- `public Treno(int[] v)`, il quale costruisce un treno formato dalla locomotiva più un numero di vagoni uguale alla lunghezza del vettore, in cui l'`i`-esimo vagone porta `v[i-1]` passeggeri;
- `public int lunghezzaTreno()`, il quale restituisce il numero (eventualmente nullo) di vagoni del treno;
- `public int personeVagone(int i)`, il quale restituisce il numero (eventualmente nullo) di passeggeri presenti nell'`i`-esimo vagone del treno;
- `public void personeCambia(int i, int p)`, il quale sostituisce col valore `p` il numero di passeggeri presenti nell'`i`-esimo vagone del treno;
- `public String toString()`, il quale stampa la struttura del treno nel formato `-|`p_1`|`p_2`|`$\cdots$`|`p_n`|`. Se, ad esempio il treno consta di 5 vagoni contenenti rispettivamente 14, 80, 4, 0, e 5 passeggeri, allora la stampa restituirà `-|14|80|4|0|5|`.

Infine, si dia un metodo `main` che a partire da un vettore intero di 7 elementi dapprima costruisce un oggetto `Treno`, poi cambia il numero di passeggeri del secondo vagone, e infine stampa l'oggetto.

Esercizio 2 Si completi la seguente classe per le liste:

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

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

in modo da aggiungere:

1. un costruttore che dato un intero `n` costruisce una lista che rappresenta la codifica binaria di `n`, con in testa il bit meno significativo di `n` (per esempio, se `n = 10` allora la lista dovrà essere `false;true;false;true` dove `true` sta per `1` e `false` sta per `0`). Si usi obbligatoriamente la ricorsione, senza né `while` né `for`;
2. un metodo `twice()` che restituisce una lista che rappresenta il doppio del numero binario rappresentato da `this` (`this` non va modificato);
3. un metodo `half()` che restituisce una lista che rappresenta la metà del numero binario rappresentato da `this` (divisione intera arrotondata per difetto, `this` non va modificato).