Tecnologie dei Linguaggi di Programmazione

Corso di Laurea in Informatica

Scritto del 14 Settembre 2010

Cognome:	Nome:	Matricola:

Esercizio 1

Dato il programma seguente:

```
class Point {
 int x;
 int y;

 public Point(int x, int y) {
      this.x = x;
      this.y = y;
 }

 static Point[] makeArray(int n) {
      Point[] res = new Point[n];
      Point pt = new Point(0,0);
      for (int i = 0; i < res.length; i++)
      {
            pt.x = i;
            res[i] = pt;
      }
      return res;
 }

 static int sum(Point[] a) {
      int res = 0;
      for (int i = 0; i < a.length; i++) {
            res += a[i].x;
      }
      return res;
 }
}
```

1. Quanto vale `Point.sum(Point.makeArray(2))` ?
2. Quanto vale `Point.sum(Point.makeArray(4))` ?
3. Quanto vale `Point.sum(Point.makeArray(6))` ?
4. Quanto vale `Point.sum(Point.makeArray(8))` ?

Esercizio 2

Date le classi seguenti:

```
class A {
  int method(B a) {
    return 2;
  }
}
class B extends A {
  int method(A a) {
    return 3;
  }
  int method(B a) {
    return 4;
  }
}
```

e gli oggetti seguenti:

```
A a1 = new A();
B a2 = new B();
A a3 = new B();
```

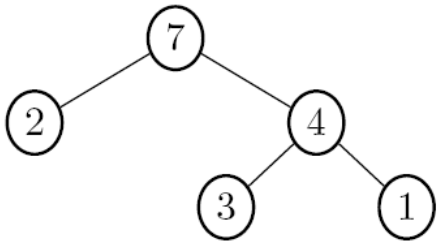
1) Quanto vale `a1.method(a1)` ?
2) Quanto vale `a1.method(a2)` ?
3) Quanto vale `a1.method(a3)` ?
4) Quanto vale `a2.method(a1)` ?
5) Quanto vale `a2.method(a2)` ?
6) Quanto vale `a2.method(a3)` ?
7) Quanto vale `a3.method(a1)` ?
8) Quanto vale `a3.method(a2)` ?
9) Quanto vale `a3.method(a3)` ?

Esercizio 3

Sia `Heap` la rappresentazione di un monticello (heap). Un elemento di tipo `Heap` è o vuoto (`Empty`) o un nodo (`Node`) composto di un valore (`val`), un sotto-monticello sinistro (`left`) e un sotto-monticello destro (`right`).

```
public class EmptyHeapException extends Exception {
}
public abstract class Heap {
}
class Empty extends Heap {
      Empty() {
      }
}
class Node extends Heap {
      int val;
      Heap left;
      Heap right;
      Node(int val, Heap left, Heap right) {
            this.val = val;
            this.left = left;
            this.right = right;
      }
}
```

Per esempio, il monticello seguente



è rappresentato come

```
Heap h = new Node(7,
      new Node(2, new Empty(), new Empty()),
      new Node(4,
            new Node(3, new Empty(), new Empty()),
            new Node(1, new Empty(), new Empty())))
```

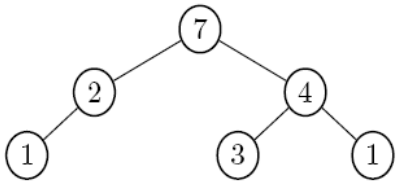
Aggiungere in `Heap`:

- un metodo pubblico `getSize` che indica il numero di nodi di un monticello. Per esempio, `h.getSize()` restituisce `5`.
- un metodo pubblico `isEmpty` che indica se un monticello è vuoto. Per esempio, `new Empty().isEmpty()` restituisce `true`.
- un metodo pubblico `isSingleton` che indica se un monticello è composto di un solo nodo. Per esempio, `new Node(2, new Empty(), new Empty()).isSingleton()` restituisce `true`.

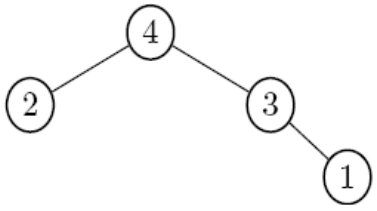
- un metodo pubblico `isWf` che indica che un monticello è *ben formato*, i.e. i numeri presenti in un qualsiasi cammino dentro il monticello dall'alto verso il basso sono in ordine decrescente. Per esempio, `h.isWf()` restituisce `true`.

Nei metodi seguenti si supponga che i monticelli manipolati siano sempre ben formati.

- un metodo pubblico `isIn` che indica se un numero è in un monticello. Per esempio, `h.isIn(2)` restituisce `true`.
- un metodo pubblico `getMin` che ritorna il valore minimo in un monticello. Tale metodo solleva l'eccezione `EmptyHeapException` se il monticello è vuoto. Per esempio, `h.getMin()` restituisce 1.
- un metodo pubblico `getMax` che ritorna il valore massimo in un monticello. Tale metodo solleva l'eccezione `EmptyHeapException` se il monticello è vuoto. Per esempio, `h.getMax()` restituisce 7.
- un metodo pubblico `add` che inserisce un nuovo valore in un monticello. Il risultato del metodo `add` è un monticello ben formato. Nell'inserimento, si aggiunge sempre nel sotto-monticello con il minore numero di nodi. Per esempio, `h.add(1)` restituisce un monticello pari a



- un metodo pubblico `remove` che elimina la radice di un monticello. Il risultato del metodo remove è un monticello ben formato. Tale metodo solleva l'eccezione EmptyHeapException se il monticello è vuoto. Per esempio, `h.remove()` restituisce un monticello pari a



Esercizio 4

Dati due array bidimensionali a e b di numeri interi non necessariamente rettangolari, definire un metodo pubblico statico `shuffle` che testa se i due array hanno gli stessi elementi, cioè l'array b si può ottenere riorganizzando opportunamente l'array a. Per esempio,

```
shuffle(new int[][] {{1, 1}, {2,2}, {1,1}}, new int[][] {{2}, {1,1,1,1}, {2}})
```

restituisce `true`. Invece

```
shuffle(new int[][] {{1, 1}, {2,2}}, new int[][] {{1,1,1}, {2}})
```

restituisce `false`.