Tecnologie dei Linguaggi di Programmazione

Corso di Laurea in Informatica

Soluzioni 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
2. Quanto vale `Point.sum(Point.makeArray(4))` ?
   12
3. Quanto vale `Point.sum(Point.makeArray(6))` ?
   30
4. Quanto vale `Point.sum(Point.makeArray(8))` ?
   56

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();
```

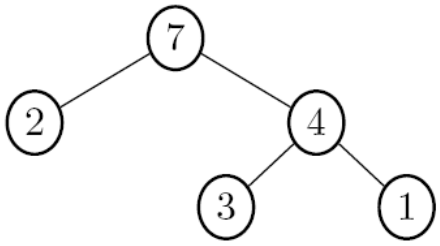
1) Quanto vale `a1.method(a1)` ?
   `NoSuchMethod`
2) Quanto vale `a1.method(a2)` ?
   2
3) Quanto vale `a1.method(a3)` ?
   `NoSuchMethod`
4) Quanto vale `a2.method(a1)` ?
   3
5) Quanto vale `a2.method(a2)` ?
   4
6) Quanto vale `a2.method(a3)` ?
   3
7) Quanto vale `a3.method(a1)` ?
   `NoSuchMethod`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

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())))
```

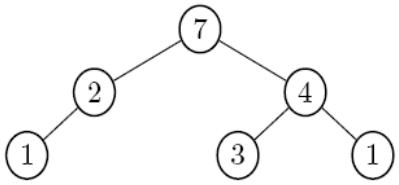
Aggiungere in `Heap`:

- 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`.

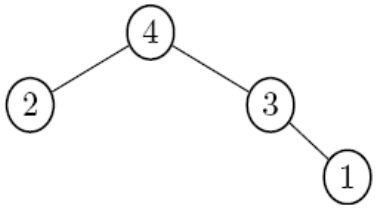
- un metodo pubblico `isWf` che indica che un monticello è *ben formato*, i.e. i numeri presenti in un qualsiasi cammino dentro il monticello dall'alto verso il basso sono in ordine decrescente. Per esempio, `h.isWf()` restituisce `true`.

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




```
public class EmptyHeapException extends Exception {
}

public abstract class Heap {
      public abstract int getSize();
      public boolean isEmpty() {
            return getSize() == 0;
      }
      public boolean isSingleton() {
            return getSize() == 1;
      }
      public abstract boolean isWf();
      abstract boolean isWf(int n);
      abstract boolean isIn(int n);
      public abstract int getMin() throws EmptyHeapException;
      public abstract int getMax() throws EmptyHeapException;
      public abstract Heap add(int n);
      public abstract Heap remove() throws EmptyHeapException;
}
```

```
class Empty extends Heap {
      Empty() {
      }
      public int getSize() {
            return 0;
      }
      public boolean isWf() {
            return true;
      };
      public boolean isWf(int n) {
            return true;
      };
      public boolean isIn(int n) {
            return false;
      }
      public int getMin() throws EmptyHeapException {
            throw new EmptyHeapException();
      }
      public int getMax() throws EmptyHeapException {
            throw new EmptyHeapException();
      }
      public Heap add(int n) {
            return new Node(n, this, this);
      }
      public Heap remove() throws EmptyHeapEcxception {
            throw new EmptyHeapException();
      }
}
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;
      }
      public int getSize() {
            return 1 + left.getSize() + right.getSize();
      }
      public boolean isWf() {
            return left.isWf(val) && right.isWf(val);
      };
      public boolean isWf(int n) {
            return (val <= n) && isWf();
      };
      public boolean isIn(int n) {
            if (val < n) {
                  return false;
            }
            return (val == n) || left.isIn(n) || right.isIn(n);
      }
      public int getMin() {
            int res = val;
            try {
                  res = left.getMin();
```

```
            } catch (EmptyHeapException e) {
            // Left is empty
            }
            try {
                  res = Math.min(res, right.getMin());
            } catch (EmptyHeapException e) {
                  // Right is empty
            }
            return res;
      }
      public int getMax() {
            return val;
      }

      public Heap add(int n) {
            if (n < val) {
                  if (left.getSize() < right.getSize()) {
                        return new Node(val, left.add(n), right);
                  }
                  return new Node(val, left, right.add(n));
            }
            if (left.getSize() < right.getSize()) {
                  return new Node(n, left.add(val), right);
            }
            return new Node(n, left, right.add(n));
      }

      public Heap remove() {
            if (left.isEmpty()) {
                  return right;
            }
            if (right.isEmpty()) {
                  return left;
            }
            Node nleft = (Node) left;
            Node nright = (Node) right;
            if (nleft.val < nright.val) {
                  return new Node(nright.val, nleft, nright.remove());
            }
            return new Node(nleft.val, nleft.remove(), nright);
      }

}
```

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`.

```
public static int occ(int[][] a, int v) {
	int res = 0;
	for (int i = 0; i < a.length; i++) {
		for (int j = 0; j < a[i].length; j++) {
			if (a[i][j] == v) {
				res++;
			}
		}
	}
	return res;
}

static boolean ishuffle(int[][] a, int[][] b) {
	for (int i = 0; i < a.length; i++) {
		for (int j = 0; j < a[i].length; j++) {
			if (occ(a, a[i][j]) != occ(b,a[i][j])) {
				return false;
			}
		}
	}
	return true;
}

public static boolean shuffle(int[][] a, int[][] b) {
	return ishuffle(a,b) && ishuffle(b,a);
}
```