Tecnologie dei Linguaggi di Programmazione

Corso di Laurea in Informatica

Soluzioni scritto del 20 Luglio 2010

Esercizio 1

Data la classe seguente:

```
class Point {
      int x, y;
      Point(int x, int y) {
            this.x = x;
            this.y = y;
      }
      void sumSwap(Point pt) {
            x = x + pt.y;
            y = y + pt.x;
      }
}
```

Dopo l'esecuzione di:

```
Point pt1 = new Point(3,4);
Point pt2 = pt1;
pt1.sumSwap(pt2);
```

Quanto vale pt1.x?
(a) 3
(b) 4
(c) 7
(d) 11

Quanto vale pt1.y?
(a) 3
(b) 4
(c) 7
(d) 11

Quanto vale pt2.x?
(a) 3
(b) 4
(c) 7
(d) 11

Quanto vale pt2.y?
(a) 3
(b) 4
(c) 7
(d) 11

Esercizio 2

Date le classi seguenti:

```
class A {
  int method(A a) {
    return 1;
  }

}

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 ();
```

2.1 Quanto vale a1.method(a1)?

2.2 Quanto vale a1.method(a2)) ?

2.3 Quanto vale a1.method(a3)) ?

2.4 Quanto vale a2.method(a1)) ?

2.5 Quanto vale a2.method(a2)) ?

2.6 Quanto vale a2.method(a3)) ?

2.7 Quanto vale a3.method(a1)) ?

2.8 Quanto vale a3.method(a2)) ?

2.9 Quanto vale a3.method(a3)) ?

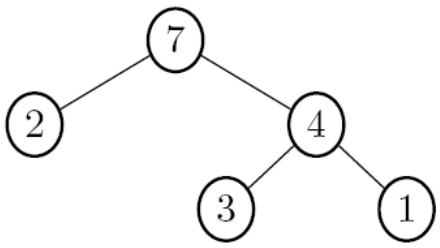
```
a1.method(a1) vale 1
a1.method(a2) vale 1
a1.method(a3) vale 1
a2.method(a1) vale 3
a2.method(a2) vale 4
a2.method(a3) vale 3
a3.method(a1) vale 3
a3.method(a2) vale 3
a3.method(a3) vale 3
```

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.

```
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;
}
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();
      }
```

```
}
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;
      }

}
```

Esercizio 4

Definire un metodo pubblico statico `copyNonZero` che prende un array bidimensionale `a` di numeri interi e restituisce un array bidimensionale `b` tale che `b` è una *quasi copia* dell'array `a`: tutti i numeri di `a` non uguali a 0 sono stati copiati in `b` e `b` non contiene nessuna riga vuota. Per esempio,

```
copyNonZero(new int[][]{{1,0,2},{0,0},{3}})
```

restituisce

```
{{1,2},{3}}.
```

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

public static int numberOfNonZeroElements(int[] a) {
	int res = 0;
	for (int i = 0; i < a.length; i++) {
		if (a[i] != 0) {
			res++;
		}
	}
	return res;
}

public static int[][] copyNonZero(int[][] a) {
	int[][] res = new int[numberOfNonZeroLines(a)][];
	int i1 = 0;
	for (int i = 0; i < a.length; i++) {
		int n = numberOfNonZeroElements(a[i]);
		if (n != 0) {
			res[i1] = new int[n];
			int j1 = 0;
			for (int j = 0; j < a[i].length; j++) {
				if (a[i][j] != 0) {
					res[i1][j1] = a[i][j];
					j1++;
				}
			}
			i1++;
		}
	}
	return res;
}
```