Tecnologie dei Linguaggi di Programmazione

Corso di Laurea in Informatica

Scritto del 30 Luglio 2010

Cognome:	Nome:	Matricola:

Esercizio 1

Data la classe seguente:

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

Dopo l'esecuzione di:

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

1. Quanto vale pt1.x?
   a) 3
   b) 4
   c) 7
   d) 11
2. Quanto vale pt1.y?
   a) 3
   b) 4
   c) 7
   d) 11
3. Quanto vale pt2.x?
   a) 3
   b) 4
   c) 7
   d) 11
4. 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;
        }
        int method(B a) {
          return 2;
        }
      }

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

class C extends B {
}
```

e gli oggetti seguenti:

```
      A a1 = new A();
      B a2 = new B();
      A a3 = new B();
      C a4 = new C();
```

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))`?
10. Quanto vale `a4.method(a4))`?

Esercizio 3

Sia `Point` la rappresentazione di un punto nel piano cartesiano.

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

Sia `Walk` la rappresentazione di un cammino nel piano cartesiano.

```
public abstract class Walk {
}

class Stop extends Walk {
}

abstract class Move extends Walk {
	Walk tail;
	Move(Walk tail) {
		this.tail = tail;
	}
}

public class Right extends Move {
	public Right(Walk tail) {
		super(tail);
	}
}

public class Left extends Move {
	public Left(Walk tail) {
		super(tail);
	}
}

public class Up extends Move {
	public Up(Walk tail) {
		super(tail);
	}
}

public class Down extends Move {
	public Down(Walk tail) {
		super(tail);
	}
}
```

tale che le seguenti dichiarazioni

```
Point p1 = new Point(0,0);
Walk w1 = new Right(new Down(new Left(new Up(new Stop()))));
Walk w2 = new Left(new Left(new Up(new Stop())));
```

creano il punto `p1` di coordinate (0,0), un cammino `w1` che va a destra, dopo va giù, quindi va a sinistra ed infine va su, ed un cammino `w2` che va a sinistra, dopo va di nuovo a sinistra ed infine va su.

Aggiungere in `Walk`:

- un metodo pubblico `isStop` che indica se un cammino non contiene alcun movimento. Per esempio, `new Stop().isStop()` restituisce `true`, invece `w1.isStop()` restituisce `false`.
- un metodo pubblico `getLength` che indica la lunghezza del cammino. Per esempio, `w1.getLength()` restituisce 4, invece `w2.getLength()` restituisce 3.
- un metodo pubblico `addTopMove` tale che `a.addTopMove(b)` restituisce un cammino che ha lo stesso primo movimento di `a` e dopo ha i movimenti di `b`. Nel caso in cui `a` è uno `Stop`, il metodo restituisce b.
- un metodo pubblico `doubleW` che crea una copia di un cammino raddoppiando ogni movimento. Per esempio, `w2.doubleW()` restituisce un cammino che va quattro volte a sinistra e due volte su.
- un metodo pubblico `concat` tale che `a.concat(b)` restituisce un cammino che inizia con i movimenti di `a` e termina con i movimenti di `b`. Per esempio, `w1.concat(w2)` restituisce un cammino che va a destra, giù, a sinistra, su, due volte a sinistra, ed infine su.
- un metodo pubblico `follow` che, dato un punto iniziale, restituisce il punto raggiunto seguendo il cammino a partire dal punto iniziale. Per esempio, `w2.follow(p1)` restituisce il punto di coordinate (-2,1).
- un metodo pubblico `isLoop` che indica se, seguendo il cammino, *alla fine* si ritorna al punto di partenza. Per esempio, `w1.isLoop()` restituisce `true` invece `w2.isLoop()` restituisce `false`.

Esercizio 4

Scrivere un metodo `isBordered` che prende un array bidimensionale quadrato di numeri interi e restituisce un booleano. Tale booleano vale `true` quando l'array è composto di bordi successivi di interi dello stesso valore, `false` altrimenti. Per esempio, dato l'array

```
{ {  4,  4,  4,  4,  4, 4  },
  {  4, -2, -2, -2, -2, 4  },
  {  4, -2,  0,  0, -2, 4  },
  {  4, -2,  0,  0, -2, 4  },
  {  4, -2, -2, -2, -2, 4  },
  {  4,  4,  4,  4,  4, 4  }  }
```

il metodo restituisce `true`.