Corso di laurea in Informatica AA 2007/08 Programmazione, M.A. Alberti

Prova scritta – II appello, 22 febbraio 2008

Cognome **Nome**

Matricola

1

Date le classi:

```
public class Padre {
  protected int n;
  protected static int x;
  Padre (int i) {
    n = i;
  }
  public void metodoA (int i) { …
  }
  public void metodoB (int i) { n=i;
  }
  public static void metodoC (int i){
    x = i
  }
}
```

```
public class Figlio extends Padre {
  Figlio (int i) { …
  }


  public static void metodoA (int i){ …
  }
  public void metodoB (int i) { n=i+1;
  }
  public static void metodoC (int i){
    System.out.println (i);
  }
}
```

Implementare il costruttore `Figlio(int i)` che inizializzi il campo `n` mediante il parametro `i`:

```
Figlio (int i) {
     super ( i );
}
```

Dire se durante la compilazione ci saranno errori e a quale dichiarazione sono dovuti:

La dichiarazione del `metodoA` della classe `Figlio` genera errore in compilazione, perchè oggetti della classe ereditano il `metodoA` dalla classe antenata e poi è dichiaratoun nuovo metodo di classe, `static`, con lo stessa firma.

Nel caso la compilazione abbia avuto successo, eventualmente eliminando la causa dei problemi, considerando le dichiarazioni seguenti:

```
Padre p = new Padre (3); Figlio f = new Figlio (7);
```

individuare quale metodo viene effettivamente invocato nel codice che segue e cosa si ottiene in output:

Eliminiamo il `metodoA` che crea problemi e di cui non viene infatti mai chiesta la valutazione

1. `System.out.println(f.n);`	7
2. `f.metodoB(4);`	–
3. `System.out.println(f.n);`	5
4. `f.metodoC(3);`	3
5. `System.out.println(f.n);`	5
6. `Figlio.metodoC(6);`	6
7. `System.out.println(p.n);`	3
8. `p.metodoB(5);`	–

```
9.  System.out.println(p.n);
10. Padre.metodoC(8);
11. System.out.println(x);
```

```
5
-
8
```

Scrivere infine l'istruzione necessaria per eseguire il `metodoC` della classe `Padre` nel contesto della definizione del `metodoC` della classe `Figlio`, nel caso in cui, ad esempio, se ne volessero semplicemente estendere le funzionalità.

```
public static void metodoC (int i){
     Padre.metodoC(i);
     System.out.println ("metodoC classe Figlio, par i: " + i);
 }
```

Rivalutare ora le istruzioni:

```
Figlio.metodoC(7);
System.out.println(x);
Padre.metodoC(8);
System.out.println(x);
```

```
7
7
-
8
```

2

Assumendo le dichiarazione seguenti:

```
final int MAX = 100;
final int MIN = 10;
int i, s=0, valore;
```

Dire qual'è il valore di `i` e `s` dopo aver eseguito:

```
if (valore <= MAX) {i = 2; if (valore > MIN) { i++; s++;}}
```

nei due casi in cui

`valore == 50` **i = 3** **s = 1**

`valore == 5` **i = 2** **s = 0**

3

Scrivere l'ouput della seguente istruzione, dopo aver eseguito l'assegnamento `numero = 3;`

```
if (numero >= 0)
     if (numero == 0) System.out.println("1");
     else System.out.println("2");
     System.out.println("3");
```

2

3

4

Assumendo la dichiarazione:

```
Random rand = new Random ();
```

Indicare il range dei valori delle seguenti dichiarazioni:

`rand.nextInt() % 30;` **[-29 29]**

`(int) (Math.random () * 5);` **[0 4]**

Inoltre scrivere un'istruzione per produrre valori paseudo-casuali nell'intervallo:

`0 – 10` usando l'oggetto `rand`

Math.abs(rand.nextInt()%11)

`10 – 20` usando il metodo `random()` della classe `Math`

(int) (Math.random()*11) + 10

5

```
public class While {
    public static void main(String[] args) {
        String originale = "catenella";
        StringBuffer copia = new StringBuffer();
        int i = 0;
        char c = originale.charAt(i);
        while (c != 'e') {
            copia.append(c);
            c = originale.charAt(++i);
        }
        System.out.println(copia);
    }
}

public class DoWhile {
    public static void main(String[] args) {
        String originale = "catenella";
        StringBuffer copia = new StringBuffer();
        int i = 0;
        char c = originale.charAt(i);
        do {
            copia.append(c);
            c = originale.charAt(++i);
        } while (c != 'e');
        System.out.println(copia);
    }
}
```

scrivere l'ouput di entrambi i cicli per la stringa `originale` data e nel caso `originale = "edizione"`

classe `While`: **cat**

classe `DoWhile`: **cat**

classe `While`: **stringa vuota**

classe `DoWhile`: **edizion**

6

Nel contesto delle seguenti dichiarazioni:

```
public class Parole {
     final char TAPPO = 0;          // TAPPO è null e indica fine stringa
     private String dato;

     Parole (String s){
          dato = s + TAPPO;
     }
     int lung () { …
     }
     String rovescia (int i) { …
     }
}
```

scrivere il metodo `rovescia`, che riporta la stringa rovesciata, in modo ricorsivo e il metodo `lung`, che calcola la lunghezza della stringa, in modo iterativo.

Suggerimento per implementare il metodo `rovescia`: se il carattere corrispondente all'indice corrente `i` è il TAPPO, allora riporta la stringa vuota altrimenti riporta la stringa ottenuta richiamando ricorsivamemnte rovescia e concatenandola con il carattere corrente. Il metodo rovescia verrà chiamato passandogli 0 come parametro.

```
String rovescia (int i) {
     if (dato.charAt(i) == TAPPO)
```

```
          return "";
     else return rovescia (i+1) + dato.charAt(i);
}

int lung() {
     int i;
     for (i=0; dato.charAt(i)!= TAPPO; i++);
     return i;
}
```

7

Date la classe seguente, che realizza una struttura FIFO (*First In First Out*):

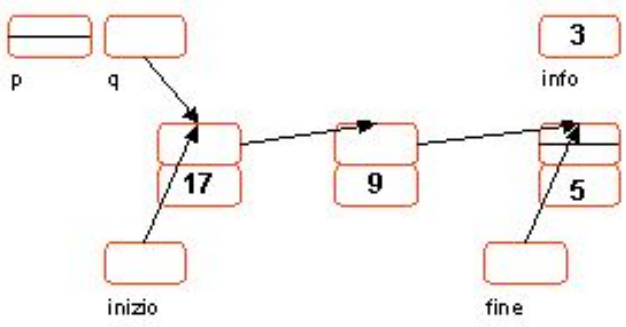
```
     public class Lista {
          private Nodo inizio, fine;
          private static class Nodo {
               int info;
               Nodo link;
          }
          Lista () {
               inizio = fine null;
          }
          void aggiungi (int n) {
               Nodo tmp = new Nodo ();
               tmp.info = n;
               tmp.link = null;
               if (inizio == null)
                   inizio = fine = tmp;
               else {
                   fine.link = tmp;
                   fine = tmp;
     0.5  0.5  }
          }
          …
     }
```

Implementare il metodo `int rimuovi()` che riporta il primo elemento eliminabile dalla lista.

```
public int rimuovi {
     if (inizio == null)
          throw new ListaVuotaException();
     else {
          int elemento = inizio.info;
          inizio = inizio.link;
          if (inizio == null)  //caso in cui la coda sia vuota
                    fine = null;
          return elemento
     }
}

public class ListaVuotaException extends RuntimeException {
}
```

8



Si usi la classe `Lista` dell'esercizio 7, per definire una lista ordinata. Si scriva il metodo `inserisci(int n)`
per effettuare l'inserimento in ordine decrescente nella lista di un nuovo nodo contenente il numero passato come.
Nella figura si suggerisce l'uso di due indici `p` e `q` che vengono inizializzati come visualizzato, per cercare la posizione corretta in cui inserire il nuovo nodo.

```
public Nodo inserisci (int n) {
     Nodo p = null;
     Nodo q = inizio;    // p e q: indici per scorrere la lista
     Nodo tmp = new Nodo ();
     tmp.info = n;
     while (p != fine && q.info > n) {
          p = q;
          q = q.link;    // si scorre tutta la lista fino alla fine
     }              // o fino a che trova un nodo con valore minore
     tmp.link = q;
     if (p != null)
          p.link = tmp;  // nuovo nodo viene riferito da p
     else inizio = tmp;  // in prima posizione aggiorniamo inizio

     if (p == fine)      // in ultima posizione aggiorniamo fine
          fine = tmp;
     return inizio;
}
```

Supponendo di avere una variabile `lista` di classe `Lista`, e una variabile `info` di tipo `int` contenente il valore numerico da inserire in lista l'istruzione sarebbe

`lista.inserisci(info)`