Corso di laurea in Comunicazione digitale — Programmazione I

AA 2009/10 — Prof. M.A. Alberti

IV Appello - 7 luglio 2010

Cognome **Nome**

Matricola

1 (6 punti)
Date le classi:

```
public class SuperCls
{
  private int n;
  protected static int x;

  SuperCls (int i) {
    n = i;
    x = i+3;
  }

  public int metodoA (int i) {
    return i;}

  public void metodoB (int i) {
    System.out.println (i);
  }

  public void metodoC (int i){
    System.out.println (i*2);
    x = i;
  }

  public static void metodD (){
…
  }
}
```

```
public class SottoCls extends
SuperCls {
  public int n;
  private int x;

  SottoCls (int i) {
    n = i;
    x = i*2;
  }

  public int metodoA(int i, int y){
    return i*y;}

  public int metodoB (int i) {
    return (i+100);
  }

  public void metodoC (int i){
    System.out.println (i);
  }


 public void metodD (int i){ …
  }
}
```

Dire se ci sono errori che impediscono la compilazione in `SuperCls`, eventualmente corregerli:

La superclasse da sola si compila correttamente. La sottoclasse contiene due errori.

il costruttore di SottoCls NON è corretto, perché manca la chiamata al costruttore della superclasse, necessario in quanto quello di default non è disponibile. La definizione del metodoB con la stessa firma ma diverso prototipo NON è possibile.

Dire se le definizioni dei membri sottoindicati della classe `SottoCls` costituiscono sovraccaricamento, sovrascrittura o errore:

`metodoA`	***sovraccaricamento***
`metodoB`	***errore***
`metodoC`	***sovrascrittura***

`metodoD` possibile ma non auspicabile. Sono metodi diversi con lo stesso nome: uno è statico e viene ereditato anche dalla sottoclasse, l'altro è un metodo d'istanza eseguibile da oggetti della sottoclasse.

Dire se e' corretta la definizione del costruttore:

```
SottoCls (int i)
```

Non e' corretta: manca l'invocazione al costruttore della super classe

Corretti gli eventuali errori, specificando quali

```
SottoCls (int i) {
        super(i);
        n = i;
        x = i*2;
    }
```

metodoB della sottoclasse si potrebbe correggere con

```
public void metodoB (int i) {
        System.out.println (i+100);
}
```

Nel metodo main della superclasse, dopo aver eseguito

```
SuperCls p = new SuperCls (3);
SottoCls f = new SottoCls (5)
```

dire quali siano i valori di ritorno o l'output di:

`p.n`	**3**
`p.x`	**8**
`f.n`	**5**
`f.metodoA (f.n, x)`	**40**
`p.x`	**8**
`p.metodoC (p.n);`	**6**
`f.metodoC (f.n);`	**5**
`p.x` **(attenzione!!)**	**3**
`f.metodoA (f.n, x)`	**15**

Dire se sia corretta l'istruzione: **NO**

```
int z = p.metodoB(15);
```

E l'istruzione: **NO**

```
int z = f.metodoB(15);
```

Spiegare:

il metodo `metodoB` della superclasse ha tipo di ritorno `void` e quindi non può essere usato per inizializzare una variabile intera.

Il metodo `metodoB` della sottoclasse per come è stato definito dà errore e quindi non può essere invocato.

---

2 (5 punti)

Scrivere la classe `Sacchetti.java` che genera 100 numeri pseudo-casusali compresi tra 1 e 12 (estremi inclusi) e che conta le occorrenze dei numeri generati tra 1 e 4, quelli tra 5 e 8 e gli altri e scriva un rapporto finale. I contatori si chiamano `sacco_1`, `sacco_2` e `sacco_3`.

```
import java.util.Random;
public class Sacchetti{
```

```
   public static void main (String[] args){
      int sacco_1=0, sacco_2=0, sacco_3=0, generato;
      Random rand = new Random ();
      for (int i=0; i < 100; i++) {
         generato = Math.abs(rand.nextInt() % 12) + 1;
         System.out.println(generato);
         if (generato <= 4)
            sacco_1++;
         else if (generato <= 8)
           sacco_2++;
         else sacco_3++;
      }
      System.out.println ("sacchetto 1: " + sacco_1);
      System.out.println ("sacchetto 2: " + sacco_2);
      System.out.println ("sacchetto 3: " + sacco_3);
 }
}
```

---

3 (4 punti)

Data la stringa di caratteri `partenza` dire l'effetto causato dall'esecuzione del ciclo sulla stringa `arrivo`:

```
String partenza = "i segni";
StringBuffer arrivo = new StringBuffer();
int i = 0;
char c = partenza.charAt(i);
while (c != 'i') {
    arrivo.append(c);
    c = partenza.charAt(++i);
}
```

Nella variabile `arrivo` alla fine del ciclo è contenuta la stringa:

""

Se, mantenendo le inizializzazioni, il ciclo fosse:

```
do {
    arrivo.append(c);
    c = partenza.charAt(++i);
} while (c != 'i');
```

la variabile `arrivo` alla fine del ciclo sarebbe:

"i segn"

Ripetere l'esercizio per la stringa `String partenza = "disegni";`

Variabile `arrivo` alla fine del ciclo while:

"d"

Variabile `arrivo` alla fine del ciclo do-while:

"d"

---

4 (4 punti)

Assumendo le dichiarazione seguenti:

```
final int MAX = 10;
final int MIN = 5;
int i, s=0, valore;
```

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

`for (i=2; valore > MIN && valore <= MAX; valore++, i++, s+=++i);`
Calcolate tre cicli `for` per diversi valori della variabile `valore`, considerandoli indipendenti l'uno dall'altro.

valore	i	s	valore
3	**2**	**0**	**3**
8	**8**	**18**	**11**
12	**2**	**0**	**12**

Se i cicli fossero eseguiti sequenzialmente otterremmo valori diversi? **SI** NO
Per quale variabile? s

valore	i	s	valore
3	**2**	**0**	**3**
8	**8**	**18**	**11**
12	**2**	**18**	**12**

---

5 (5 punti)
Scrivere una funzione **ricorsiva** per il calcolo del numero delle cifre in un numero naturale. Si ricorda che se il numero dato è compreso tra 0 e 9 allora è composta da 1 cifra. Nel caso di numeri più lunghi si potrà procedere con successive divisioni per 10 .

```
int numero_cifre(int n) {
    if (n>=0 && n<=9)
            return 1;
        else
        return 1 + numeroCifre(n/10);   // indirizzo A
}
```

---

6 (6 punti)
Disegnate la traccia dell'andamento dello stack delle chiamate dei metodi durante l'esecuzione del metodo **`int numero_cifre(int n)`** con valori `numero_cifre(2010)`. Si ricordi che, nel record d'attivazione del metodo i campi `ind` e `val` indicano rispettivamente l'indirizzo e il valore di rientro relativi ad un metodo chiamato. Inoltre si noti che nella figura è tracciata solo la parte di stack relativa alla prima chiamata del metodo `f` dall'inidirizzo `K` di qualche altro metodo che trascuriamo. La freccia indica la direzione di crescita dello stack

	n 2010	n 201	n 20	n 2
val ~~?~~ 4	val ~~?~~ 3	val ~~?~~ 2	val ~~?~~ 1	val ?
ind K	ind ~~?~~ A	ind ~~?~~ A	ind ~~?~~ A	ind ?

Record 1 Record 2 →

---

7 (4 punti)
Utilizzando la classe `StringTokenizer` del pacchetto `java.util` scrivete un ciclo `while` per testare se ci sono ancora token e stamparli. Inizializzate l'oggetto `st` di classe `StringTokenizer` con la stringa `"questo e' un esame"`

```
StringTokenizer st = new StringTokenizer("questo e' un esame");

while (st.hasMoreTokens()) {
```

```
        System.out.println(st.nextToken());
}
```

Dire qual'è il tipo riportato da `nextToken()`: **`String`**