

Cognome

Nome

Matricola

1

Data la classe `Padre` e la sua sottoclasse `Figlio`:

<pre>public class Padre { protected static int m; private int n; Padre() { m++; n = 5; } protected int getn() { return n; } public int metodo_1(int i) { return n=i; } protected void metodo_2(int i) { n=i+2; } public static void metodo_3(int i) { stampa(i+3); } public static void main(...) { Padre p = new Padre(); int x = 8; stampa(p.metodo_1(1)); p.metodo_2(2); stampa(p.getn()); Padre.metodo_3(x++); stampa(Padre.m); } }</pre>	<pre>class Figlio extends Padre { static int n=20; int m; Figlio() { super(); } public int getm() { return m; } public int metodo_1(int i) { return m += i; } public void metodo_2(double i) { m=(int)i*2; } public static void metodo_3(float i){ } public static void metodo_4 (int i, int j) { stampa(n+i+j); n-- } }</pre>
---	--

Per ciascun metodo della sottoclasse `Figlio` dire se la definizione causa *errore* in compilazione e nel caso, specificare quale, o dire se si tratta di *sovrascrittura* o di *sovraccaricamento* o di *specializzazione*.

metodo_1: **sovrascrittura**

metodo_3: **sovraccaricamento**

metodo_2: **sovraccaricamento**

metodo_4: **specializzazione**

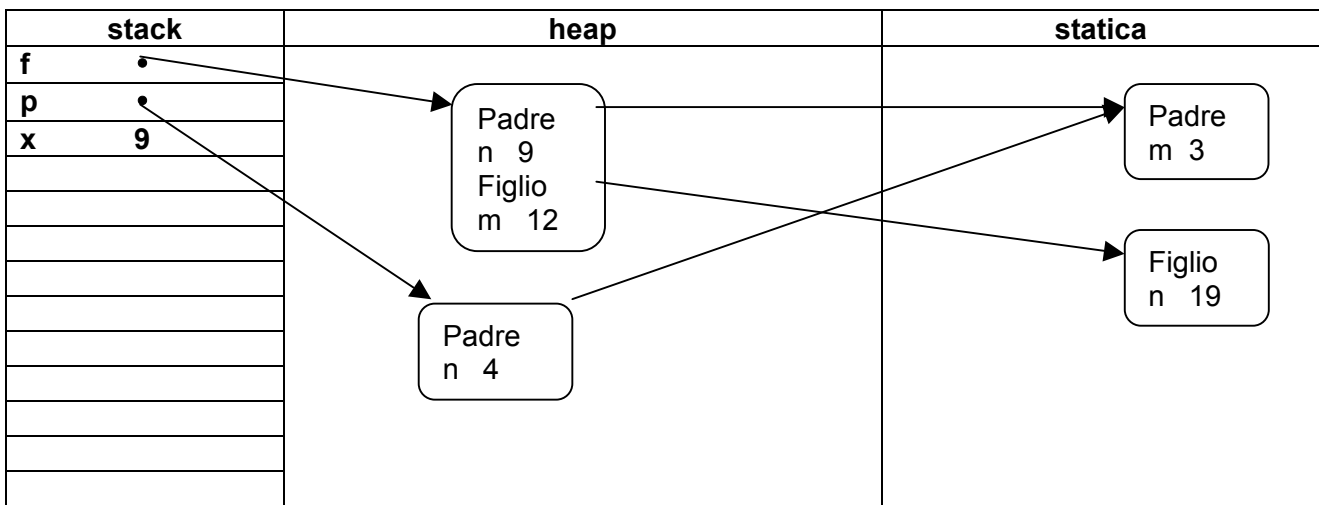
Definire il costruttore `Figlio()`.

Nel contesto della classe `Figlio`, dopo aver eseguito: `Figlio f = new Figlio();` valutare le istruzioni:

1. <code>stampa(f.getn());</code>	5		2
2. <code>Padre.main(null);</code>	1	3. <code>stampa(f.metodo_1(1));</code>	1
	4	4. <code>stampa(f.getm());</code>	1
	11	5. <code>f.metodo_2(6.5);</code>	

6. <code>stampa(f.getm());</code>	12	11. <code>stampa(f.getn());</code>	9
7. <code>Figlio.metodo_4(2,2);</code>	24	12. <code>Padre.main(null);</code>	1
8. <code>stampa(Figlio.n);</code>	19		4
9. <code>f.metodo_2(7);</code>			11
10. <code>Figlio.metodo_3(3);</code>	6		3

Disegnate lo stato dello stack, dello heap e della memoria statica (per le sole parti rilevanti al fine di poter valutare il valore dei campi di classe e di istanza), durante l'esecuzione dell'istruzione 12., in particolare dopo aver eseguito l'ultima istruzione, immediatamente prima di concludere il metodo `main` della classe `Padre`.



2

Considerate le classi A e B, estensione della classe A:

<pre>class A { public int codice; protected String tipo; A(int c, String t) { codice = c; tipo = t; } }</pre>	<pre>class B extends A { private int codice; B(int c) { super(c-1, "esteso"); codice = c; } }</pre>
--	--

- Definite il costruttore della classe B. Ricordate che la classe B è un'estensione della classe A, quindi nel definire il costruttore utilizzate opportunamente le seguenti indicazioni:
 - ogni oggetto di classe B è caratterizzato da un numero di codice, archiviato nel campo `codice`
 - il suo antenato ha per convenzione il campo `codice` il cui valore è dato dal valore - 1 del codice del figlio e il campo `tipo` posto al valore predefinito "esteso".
- Osservate i due membri `codice` definiti nelle due classi. Il membro `codice` di classe A dichiarato `public` viene ereditato dalla classe B? SI **NO**
- Si dice che il membro `codice` di classe B **oscura** quello di classe A SI **NO**
- Dire se l'istruzione seguente, che definisce un metodo di classe B, è corretta: **SI** **NO**
`int get_codice() { return super.codice + codice; }`
- Definita nella classe A un metodo `main` con le istruzioni seguenti, calcolate il valore che viene stampato:

<pre> public static void main (String[] s) { A a = new A(21, "non esteso"); B b = new B(222); System.out.println(a.codice); System.out.println(a.tipo); System.out.println(b.get_codice()); System.out.println(b.tipo); } </pre>	<pre> 21 non esteso 443 esteso </pre>
--	---------------------------------------

3

Scrivere un programma per computare la famosa successione di Fibonacci $F(n)$ data dai numeri 1, 1, 2, 3, 5, 8, 13, 21, ... La sequenza è così definita:

$F(0) = 1, F(1) = 1, F(2) = F(1) + F(0), F(3) = F(2) + F(1) \dots$

Il programma, dato il parametro n dovrà riportare il valore del n -esimo numero della sequenza di Fibonacci.

Versione ricorsiva:

```

public static int fib(int n) {
    if (n==0 || n==1)
        return n;
    else return fib(n-1)+fib(n-2);
}

```

Versione iterativa:

```

public static int fibIt(int n) {
    int prec = 1, precPrec = 1, risultato = 1;
    if (n==0 || n==1)
        return risultato;
    else {
        for (int i=2; i<=n; i++) {
            risultato = prec + precPrec;
            precPrec = prec;
            prec = risultato;
        }
        return risultato;
    }
}

```

Calcolare il valore di $F(5)$: **8**

Dire il numero di chiamate ricorsive effettuate per calcolare $F(5)$: **15**

4

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 = new String (s) + TAPPO;
    }
    int lung ()    {
        int i; char c;
        for (i=0; dato.charAt(i) != TAPPO; i++);
        return i;
    }
    String rovescia (int i) {
        if (dato.charAt(i) == TAPPO)
            return "";
        else return rovescia (i+1) + dato.charAt(i);
    }
}

```

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

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 opportunamente il metodo `rovescia` e concatenando il risultato con il carattere corrente. Il metodo `rovescia` verrà inizialmente chiamato passandogli 0 come parametro.

5

Data la frase "Sempre caro mi fu quest'ermo colle", scrivere un metodo `InputInArray` che riceve la stringa come parametro e usa la classe `StringTokenizer` per selezionarne le singole parole e archivarle in array di lunghezza appropriata da riportare all'ambiente chiamante.

Prototipo:

```
String[] inputInArray(String)
```

Codice

```
public static String[] inputInArray (String s) {
    StringTokenizer st = new StringTokenizer(s);
    String[] struttura = new String[st.countTokens()];
    for (int i=0; st.hasMoreTokens(); i++) {
        struttura[i]=st.nextToken();
    }
    return struttura;
}
```

6

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

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

la stringa `arrivo` alla fine del ciclo: **mi**

E nel caso il ciclo fosse sostituito da un simile ciclo `do-while`?

```
do {
    copia.append(c);
    c = originale.charAt(++i);
} while (c != 'a');
```

la stringa `arrivo` alla fine del ciclo: **mi**

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

la stringa `arrivo` alla fine del ciclo `while`: **null**

la stringa `arrivo` alla fine del ciclo `do-while`: **anfor**