A

Esame di Linguaggi di Programmazione (proff. Picco/Ronchetti)
Prova scritta – 11 settembre 2018

Istruzioni (leggere attentamente!)

Staccare questo foglio dal resto del blocchetto, ed utilizzarlo per segnare le risposte sul retro di questa pagina. Scrivere subito nome, cognome e numero di matricola negli appositi spazi. Al termine della prova si dovrà consegnare SOLO questo foglio; potete tenere gli altri.

Per le domande che includono codice Java, quando non vengono mostrate le istruzioni **`import`** queste si assumono essere correttamente presenti.

Vi sono tre tipi di domande e, di conseguenza, risposte:

1) *Un frammento di codice che esegue correttamente*. Si indichi l'output

TEST 1	l'output è →	

2) *Un frammento di codice che genera errori*. Si indichi in quale riga l'errore avviene; si specifichi la tipologia (in compilazione o a runtime) cerchiando la cella corrispondente (A o B) e si spieghi brevemente la ragione dell'errore.

TEST 2	errore alla riga _____	A	in compilazione	perché →	
		B	a run time		

3) *Domande vero/falso*. Si riporti V o F nelle caselle corrispondenti. In questa tipologia, le risposte errate sottraggono punti. Nel dubbio si consiglia di lasciare in bianco.

A

NOME, COGNOME	
NUMERO DI MATRICOLA	
CORSO DI LAUREA	

TEST 1	l'output è →	

TEST 2	l'output è →	

TEST 3	errore alla riga _____	A	in compilazione	perché →	
		B	a run time		

TEST 4	l'output è →	

TEST 5	errore alla riga _____	A	in compilazione	perché →	
		B	a run time		

TEST 6	errore alla riga _____	A	in compilazione	perché →	
		B	a run time		

TEST 7	l'output è →	

TEST 8	errore alla riga _____	A	in compilazione	perché →	
		B	a run time		

TEST 9	Indicare, per ogni sotto-domanda contrassegnata dalla lettera corrispondente se la risposta è V (vero) o F (falso) →	A	B	C	D
		E	F	G	H

A

Test 1

```
01   #include <cstdlib>
02   #include <iostream>
03   using namespace std;
04   int x[] = {2, 7, 3, 8};
05   int f(int *x, int *y){
06     y[3] = 5;
07     return ++(*x);
08   }
09   int g(int a){
10     x[0] = 0;
11     return f(&x[a], x) ;
12   }
13   int main() {
14       int a = 1;
15       x[a] = a;
16       cout << g(++a);
17       for (int i = 0; i<4; i++) cout << x[i];
18       return 0;
19   }
```

Test 2

```
01   public class Test2 {
02     public static void main(String[] args) {
03       A[] a = new A[4];
04       for (int i = 0; i < a.length; i++) {
05         if (i%2 == 0) a[i]= new A(i);
06         else a[i]= new B(i);
07       }
08       ArrayList<A> l = new ArrayList<>(Arrays.asList(a));
09       for(A e: l) System.out.print(e.m(5) + " ");
10     }
11   }
12   class A {
13     int x;
14     A(int x) { this.x = x + 1; }
15     public int m(int z) { return x + z; }
16   }
17   class B extends A {
18     B(int x) { super(x); }
19     public int m(int z) { return super.m(z) * 2; }
20   }
```

Test 3

```
00   public class Test3 {
01     public static void main(String[] args) {
02       A obj = new B();
03       obj.m(new D());
04     }
05   }
06   class A {
07     final void m(C c) { System.out.println("1"); }
08   }
09   class B extends A {
10     void m(C c) { System.out.println("2"); }
11     void m(D c) { System.out.println("3"); }
12   }
13   class C {}
14   class D extends C {}
```

A

Test 4

```
01    public class Test4 {
02      public static void main(String[] args) {
03        I i = new C(3);
04        System.out.println(i.m(5));
05      }
06    }
07    interface I {
08      int m(int z);
09    }
10    class A implements I {
11      int x;
12      A(int x) { this.x = x + 1; }
13      public int m(int z) { return x + z; }
14    }
15    class B extends A {
16      B(int x) { super(++x); }
17      public int m(int z) { return x * z; }
18    }
19    class C extends B {
20      C(int x) { super(++x); }
21    }
```

Test 5

```
01    public class Test5 {
02      public static void main(String[] args) {
03        B b = new B();
04        A a = new A();
05        J j = b;
06        System.out.println(j.m(5) + b.m("hello"));
07      }
08    }
09    interface I {
10      int m(int z);
11    }
12    interface J extends I {
13      int m(String s);
14    }
15    class A implements I {
16      int x = 5;
17      public int m(int z) { return x + z; }
18    }
19    class B implements J {
20      public int m(String s) { return s.length(); }
21    }
```

A

Test 6

```
01  public class Test6 {
02    static final ArrayList<A> l = new ArrayList<>();
03    void doSomething(int x) {
04      for(int i=0; i<x; i++) {
05        B b = new B();
06        b.m();
07      }
08    }
09    void doSomethingElse() {
10      for(int i=0; i<l.size(); i++) {
11        A a = l.get(i);
12        a++;
13      }
14    }
15    public static void main(String[] args) {
16      Test6 t = new Test6();
17      t.doSomething(5);
18      A a = new A();
19      t.doSomethingElse();
20    }
21  }
22  class A {
23    A() { Test6.l.add(this); }
24  }
25  class B extends A {
26    int x = 0;
27    B() { super(); }
28    void m() { x++; }
29  }
```

Test 7

```
01   public class Test7 {
02     public static void main(String[] args) {
03       A a = new A();
04       a.m1(a.y);
05       System.out.println("y="+a.y);
06     }
07   }
08   class A {
09     static int y = 1;
10     void m1(int y) {
11       y++;
12       m2();
13     }
14     void m2() { ++y; }
15   }
```

A

Test 8

```
01  public class Test8 {
02    public static void main(String[] args) {
03      B b1 = new B(new A());
04      b1.m(3);
05      B b2 = new B();
06      b2.m(5);
07      System.out.println(b2);
08    }
09  }
10  class A {
11    static int x = 3;
12    A() { x++; }
13    void m(int x) { this.x += x; }
14    public String toString() { return "x="+x; }
15  }
16  class B {
17    A a;
18    B(A a) { this.a = a; }
19    B() { super(); }
20    void m(int x) { a.m(x); }
21    public String toString() { return a.toString(); }
22  }
```

Test 9

A	Una classe Java definita come **`abstract`** può essere usata all'interno di gerarchie di classi con ereditarietà multipla
B	Il tipo array **`int[]`** è un tipo riferimento, come le classi.
C	Un metodo generico è un qualsiasi metodo che contiene parametri di tipo **`Object`** nella propria definizione.
D	Sia data una classe Java **`A`** che contiene un metodo **`m()`**. Una sottoclasse **`B`** di **`A`** può ridefinire **`m()`** una sola volta mediante *override* e un numero arbitrario di volte mediante *overload*.
E	In Java esiste ereditarietà singola: quindi, un tipo interfaccia può ereditare da un solo tipo interfaccia.
F	In Java, se **`B`** è una sottoclasse di **`A`** l'istruzione **`B a = new A();`** genera errore a runtime.
G	Con il termine "*autoboxing*" in Java ci si riferisce alla capacità di una finestra grafica di adattare automaticamente la propria dimensione (in pixel) al proprio contenuto.
H	La parola chiave Java **`this`** serve a identificare un particolare elemento all'interno di un array.