A

Università degli Studi di Trento — 5 febbraio 2020

Linguaggi di Programmazione — Programmazione 2
Linguaggi di Programmazione (Mod. 1)

Proff. Picco e Ronchetti — *Prova scritta*

Istruzioni (leggere attentamente)

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

1. **Un frammento di codice che esegue correttamente.** Si indichi l'output nell'apposito riquadro.

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); si spieghi brevemente la ragione dell'errore. Tutti e tre i riquadri devono essere compilati.

Test 2	errore alla riga n. ———	A	in compilazione	perché →	
		B	a runtime		

3. **Domande vero/falso.** Si riporti `V` (vero) o `F` (falso) nelle caselle corrispondenti. In questa tipologia, *le risposte errate sottraggono punti.*

Test 3	A	B	C	D	E	F	G	H

*Staccare questo foglio dal resto del blocchetto: 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.*

A

Nome e cognome	
Numero di matricola	
Corso di Studi	☐ INF ☐ ING-ORG ☐ MAT ☐ altro

Test 1	errore alla riga n. ———	A	in compilazione	perché →	
		B	a runtime		
Test 2	errore alla riga n. ———	A	in compilazione	perché →	
		B	a runtime		
Test 3	errore alla riga n. ———	A	in compilazione	perché →	
		B	a runtime		

Test 4	l'output è →	
Test 5	l'output è →	
Test 6	l'output è →	
Test 7	l'output è →	
Test 8	l'output è →	

Test 9	A	B	C	D	E	F	G	H

spazio per il docente: non compilare!				
codice	vero/falso			voto finale
	corrette	errate	punti	

A

Test 1

```
1  public class Test {
2    static final int MAX = 10;
3    public static void main(String[] args) {
4      A a = new A();
5      a.m1(1);
6      C c =  new C();
7      c.m2(2);
8    }
9  }
10 interface I1 {
11   public int m1(int x);
12 }
13 interface I2 extends I1 {
14   public int m2(int x);
15 }
16 class A implements I1 {
17   int a = 20;
18   public int m1(int x) {
19     return a + x;
20   }
21 }
22 class B extends A {}
23 class C extends B implements I2 {}
```

Test 2

```
1  public class Test {
2    static final int MAX = 10;
3    public static void main(String[] args) {
4      A a = new A(10);
5      a.m("mela");
6      A a1 = new C();
7      C c1 = (C) a1;
8      c1.m("pera");
9      A a2 = new B();
10     C c2 = (C) a2;
11     c2.m("banana");
12   }
13 }
14 interface I1 {
15   public int m(String s);
16 }
17 class A implements I1 {
18   int x;
19   A(int x) { this.x = x; }
20
21   public int m(String s) {
22     return s.length() * x;
23   }
24 }
25 class B extends A {
26   B() { super(0); }
27 }
28 class C extends B {}
```

A

Test 3

```
1  public class Test {
2    public static void main(String[] args) {
3      A obj = new B();
4      obj.m(new D());
5    }
6  }
7  class A {
8    final void m(C c) { System.out.println("1"); }
9  }
10 class B extends A {
11   void m(C c) { System.out.println("2"); }
12   void m(D c) { System.out.println("3"); }
13 }
14 class C {}
15 class D extends C {}
```

Test 4

```
1  public class Test {
2    public static void main(String[] args) {
3      A obj = new B();
4      A par = new B();
5      System.out.println(obj.m(par));
6    }
7  }
8  class A {
9    int x = 0;
10   String m(A a) { return "a␣in␣a"; }
11 }
12 class B extends A {
13   String m(B b) { return "b␣in␣b"; }
14   String m(A a) { return "a␣in␣b"; }
15 }
```

Test 5

```
1  public class Test {
2    public static void main(String[] args) {
3      A a1 = new A(5);
4      A a2 = new A(3);
5      A a3 = new A(5);
6      System.out.println(a1.equals(a2));
7      System.out.println(!a1.equals(a3));
8    }
9  }
10 class A {
11   int x = 0;
12   A(int x) { this.x = x; }
13 }
```

A

Test 6

```
1  public class Test {
2    public static void main(String[] args) {
3      A a = new A();
4      int r1 = a.m(6);
5      A a2 = new B(new Z());
6      int r2 = a2.m(1);
7      B b = new B(new Z());
8      int r3 = b.m(2);
9      System.out.println("" + r1 + "␣" + r2 + "␣" + r3);
10   }
11 }
12 class Z {}
13 class A {
14   static int val = 5;
15   int m(int x) { val--; return x * val; }
16 }
17 class B extends A {
18   Z z;
19   B(Z z) { this.z = z; val++; }
20   int m(int x) { return x * val + 1; }
21 }
```

Test 7

```
1  public class Test {
2    public static void main(String[] args) {
3      I i = new C(4);
4      System.out.println(i.m(2));
5    }
6  }
7  interface I {
8    int m(int z);
9  }
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 8

```
1  import java.util.*;
2  public class Test {
3    public static void main(String[] args) {
4      String[] a = {"limone", "ananas", "mango", "lime"};
5      Set<Integer> s = new HashSet<>();
6      List<Integer> l = new ArrayList<>();
7      for(String i: a) {
8        s.add(i.length());
9        l.add(i.length());
10     }
11     System.out.println(s.size() + l.size());
12   }
13 }
```

Test 9

A	In una classe possono coesistere più metodi con lo stesso nome ma firme (numero e tipo dei parametri) diverse.
B	La parola chiave `finally` serve a terminare l'esecuzione del programma.
C	Si consideri un attributo `x` dichiarato come `protected` nella classe `C` del package `P1`; `x` non è visibile da una classe `D` appartenente a un package `P2`, a meno che `D` non erediti da `C`.
D	Le istruzioni `Object[] x = new Object[10]; x[0] = "elemento";` sono illegali in Java, perché per gli array non è supportato il polimorfismo; questo è il motivo principale per l'esistenza del Java Collection Framework, che invece supporta questa funzionalità.
E	Il *garbage collector* è una funzionalità del supporto run-time di Java molto utile, in quanto gestisce in maniera automatica la deallocazione di oggetti non più utilizzati dal programma, semplificando il lavoro del programmatore ed evitandone potenziali errori.
F	Quando in Java si parla di un metodo *overloaded* si fa riferimento a un metodo "sovraccarico", cioè la cui implementazione genera una computazione particolarmente pesante.
G	Una classe Java definita come `abstract` può essere usata all'interno di gerarchie di classi con ereditarietà multipla.
H	Siano dati due oggetti `a` e `b`, per i quali il test `a.hashCode() == b.hashCode()` ritorna `true`. In questo caso, si può stabilire con certezza che `a` e `b` sono *identici*, cioè puntano allo stesso oggetto.