A

Test 1

```
#include <cstdlib>
#include <iostream>
using namespace std;
int x = 11;
int f(int *x){ return ++(*x); }
int g(int y){  return f(&y) + f(&x); }
int main() {
    int a = 3;
    cout << g(a);
    cout << a;
    cout << x;
    return 0;
}
```

Test 2

```
public class Test2 {
  public static void main(String[] args) {
    A a = new B(3);
    C c = new C(3);
    System.out.println(a.m() + c.m());
  }
}
class A  {
  private int x;
  int m() { return x + 1; }
}
class B extends A {
  B(int x) { this.x = x; }
  int m() { return x + 2; }
}
class C {
  int x = 0;
  A a = new A();
  C(int x) { this.x = a.x; }
  int m() { return x + 3; }
}
```

Test 3

```
public class Test3 {
  public static void main(String[] args) {
    A obj = new B();
    obj.m(new D());
  }
}

class A {
  void m(C c) { System.out.println("1"); }
}
class B extends A {
  void m(C c) { System.out.println("2"); }
}
class C extends B {}
class D extends A {}
```

A

Test 4

```
01  public class Test4 {
02    public static void main(String[] args) {
03      A a = new B(7);
04      I i = a;
05      System.out.println(i.m(4));
06    }
07  }
08  interface I {
09    int m(int z);
10  }
11  class A implements I {
12    int x;
13    A(int x) { this.x = x - 1; }
14    public int m(int z) { return x + z; }
15  }
16  class B extends A {
17    B(int x) { super(x); }
18    public int m(int z) { return x - z; }
19  }
```

Test 5

```
01  public class Test5 {
02    public static void main(String[] args) {
03      B b = new B();
04      A a = b;
05      J j = a;
06      System.out.println(j.m(5) + b.m("ciccio"));
07    }
08  }
09  interface I {
10    int m(int z);
11  }
12  interface J extends I {}
13
14  class A implements I {
15    int x = 3;
16    public int m(int z) { return x + z; }
17  }
18  class B extends A implements J {
19    public int m(String s) { return s.length(); }
20  }
```

A

Test 6

```
01   public class Test6 {
02     static final int N = 10;
03     public static void main(String[] a) {
04       Test6 t = new Test6();
05       t.m(N);
06     }
07     void m(int n) {
08       List<Integer> la = new ArrayList<>();
09       List<Integer> lb = new ArrayList<>();
10       for(int i=0; i<N; i++)
11         la.add(i);
12       for(int i=0; i<2*N; i++)
13         if (i%2==0) la.add(i);
14       la.addAll(lb);
15       System.out.println(la.size());
16       Set<Integer> s = new HashSet<>();
17       s.addAll(la);
18       System.out.println(s.size());
19     }
20   }
```

Test 7

```
01   public class Test7 {
02     public static void main(String[] args) {
03       List<A> ls = new ArrayList<>();
04       ls.add(new A("banana")); ls.add(new A("kiwi")); ls.add(new A("avocado"));
05       Collections.sort(ls);
06       for(A s: ls)
07         System.out.println(a.s.length());
08     }
09   }
10   class A implements Comparable<A> {
11     String s;
12     A(String s) { this.s = s; }
13     public int compareTo(A a) {
14       return (s.length() - a.s.length());
15     }
16   }
```

Test 8

```
01  public class Test8 {
02    public static void main(String[] args) {
03      A a1 = new A(5);
04      A a2 = new A(a1.m()/2);
05      boolean b = (a1 == a2);
06      System.out.println(b);
07
08    }
09  }
10  class A  {
11    int x;
12    A(int x) { this.x = x; }
13    int m() { return x*2; }
14  }
```

A

Test 9

A	In una classe possono coesistere più metodi con lo stesso nome ma firme (numero e tipo dei parametri) diverse.
B	Il costruttore di una classe chiama automaticamente il costruttore della superclasse con gli stessi parametri; se in questa non è disponibile un costruttore con la stessa firma, viene chiamato il costruttore senza parametri.
C	Sia `I` un'interfaccia, e `C` una classe che la implementa. L'assegnamento `C c = new I();` non causa errori in compilazione, ma può generare errori a runtime.
D	Un attributo `x` dichiarato come `public` nella classe `C` del package `P1` è visibile a tutte le altre classi dichiarate in `P1`, ma non a classi dichiarate in package diversi da `P1`.
E	Un'interfaccia Java normalmente contiene solo la firma dei metodi; la loro definizione (corpo) nell'interfaccia è opzionale.
F	Java non ha un distruttore (simile alla `destroy` del C++) per deallocare esplicitamente e direttamente oggetti in memoria.
G	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.
H	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.