C

Test 1

```
01  #include <iostream>
02  using namespace std;
03  void g(char x[], int y) {
04     y--;
05     x[y]--;  }
06  void f(char *x, int * y) {
07     (*y)++;
08     x[*y]++;  }
09  int main(){
10    char x[2];
11    int y;
12    x[0]='D';  x[1]='D';   y=0;
13    f(x,&y);
14    g(x,y);
15    cout<<x[0]<<" "<<x[1]<<" "<<y;
16    return 0; }
```

CE1 √

Test 2 java F

```
01  public class F{
02      int x=3;
03      F(int x) {
04         f(x);
05         f();
06         System.out.println(x);
07      }
08      void f() { x++; System.out.print(x);}
09      void f(int x) { this.x++; x--;System.out.print(x);}
10      public static void main(String arg[]) {
11            F x=new F(9);
12      }}
```

84
859

Test 3 java G

```
01  public class G implements Cloneable{
02      int k=0;
03      public G clone() {
04          G copia=null;
05          try {
06              copia=(G)super.clone();
07          }catch(CloneNotSupportedException e){System.exit(0);}
08          copia.k++;
09          return copia;
10      }
11      public int hashCode(){return 0;}
12      public boolean equals(Object x) {
13          if (! (x instanceof G)) return false;
14          return k==((G)x).k;
15      }
16      public static void main(String[] args) {
17          G b= new G();
18          G c=(G)(b.clone());
19          G d=new G();
20          if (b.equals(c)) System.out.print("D");
21          if (c.equals(d)) System.out.print("E");
22          if (d.equals(b)) System.out.print("F");
23      }
24  }
```

EF

F

C

Test 4 java C

```
00  class C{
01      public static int x;
02      C(int s) {x=s;}
03      void f() {System.out.print(x);}
04      public static void main(String a[]){
05          C b=new C(7);
06          C c=new C(10);
07          b.f();
08          c.f();
09   }   }
```

10 10

Test 5 java B

```
01  import java.util.*;
02  public class B {
03      B(){
04          Collection b = new Collection();
05          for (int k=0;k<10;k++) {
06              String s="A"+(k%4);
07              b.add(s);
08          }
09          int count=0;
10          Iterator i=b.iterator();
11          while (i.hasNext()) {
12              Object s=i.next();
13              count++;
14          }
15          System.out.println(count);
16      }
17      public static void main(String[] a) { new B();new B(); }
```

ERR ←
compile

Test 6 java A

```
00  import java.util.*;
01  public class A {
02      A(int m){
03          List<String> b = new TreeSet<String>();
04          for (int k=0;k<10;k++) {
05              String s="A"+(k%m);
06              b.add(s);
07          }
08          int count=0;
09          Iterator<String> i=b.iterator();
10          while (i.hasNext()) {
11              String s=i.next();
12              System.out.print(s);
13          }
14      }
15      public static void main(String[] a) { new A(3); }}
```

? ←

012

C

Test 7: java E

```
01  class E {
02      static int s=0;
03      E(int i){s=i;}
04      public static void main(String[] args) {
05          E b1=new E(3);
06          E b2=new E(3);
07          E b3=new E(1);
08          if (b1.equals(b2)) System.out.print("K"); else
    System.out.print("X");
09          if (b1.s==b3.s) System.out.print("IA"); else
    System.out.print("D");
10  }   }
```

X IA

Test 8 java D

```
01  public class D { static int x=1; S5 a=null;
02      class S5 { int k;
03          S5() {k=x;}
04          public void finalize() { System.out.print(k); }
05      }
06      D() {x++; a=new S5();}
07      void f() {S5 a=new S5();}
08      public void finalize() { System.out.print("1"); }
09      public static void main(String[] args) {
10          D a5=new D();       a5.f();
11          a5=new D();         a5.f();
12          System.gc();        System.runFinalization();
13          System.out.print("-");
14          a5=null; System.gc();       System.runFinalization();
15      }
16      public static void main(String args) { D a5=new D();}
17  }
```

31

3122-31

Test 9 – scrivere nel campo per l'output del test la sequenza risultante indicando V per le affermazioni vere e F per quelle false

V	9.1	Un oggetto ed un suo clone non sono identici.
F	9.2	Poichè Java usa sempre dynamic binding, esso usa sempre la heap e mai lo stack.
F	9.3	Ereditarietà multipla è permessa con le interfacce e le classi astratte.
V	9.4	Di default l'operatore = = e il metodo equals fanno la stessa cosa.
F	9.5	Il main può accedere a qualunque variabile di istanza della classe in cui è contenuto.
V	9.6	Il costruttore di una classe può non essere visibile all'esterno della classe stessa.
F	9.7	Se B estende A la scrittura B a=new A(); genera errore a run time
F	9.8	Se A è padre di B la scrittura B a=(B)(new A()); genera errore a run time

6 1/2 / 9