A

Test 1 java Main

```
00 class A {
01 A(){System.out.print("A");}
02 A(int i) {System.out.print("A"+i);}
03 public void finalize() {System.out.print("Z");}
04 }
05 class B extends A{
06 B(){System.out.print("B");}
07 B(int i) {System.out.print("B"+i);}
08 public void finalize() {System.out.println("W");}
09 }
10 public class Main {
11     public static void main(String[] args) {
12      new B(3);
13      System.gc();
14     }
15 }
```

Test 2

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

Test 3 java Main

```
01 class B {
02     int s=0;
03     B(int s) {this.s=s;}
04     public boolean equals(Object o) {
05         return (s==((B)o).s);
06  }   }
07 class A extends B {
08     static int w=0;
09     A() {
10         super(++w);
11  }   }
12 public class Main {
13     public static void main(String[] args) {
14      B b1=new B(2);   B b2=new A();   A a=new A();
15      if (b1.equals(b2)) System.out.print("A"); else
   System.out.print("B");
16      if (b1.equals(a)) System.out.print("C"); else
   System.out.print("D");
17  }   }
```

A

Test 4 java B

```
00 class B {
01     int s=0;
02     B(int i){s=i;}
05     public static void main(String[] args) {
06      B b1=new B(3);
07      B b2=new B(3);
08      B b3=new B(1);
09      if (b1.equals(b3)) System.out.print("A"); else
System.out.print("B");
10      if (b2.equals(b3)) System.out.print("C"); else
System.out.print("D");
11    }
12 }
```

Test 5 java Main

```
01 interface A {
02     int f(int x);
03 }
04 class B implements A {
05     public int f(int w){
06         return w*2;
07  } }
08 class C extends B {
09     int f(float w){
10         return (int)(w*4);
11 } }
12 public class Main {
13     public static void main(String[] args) {
14         A a=new C();
15         System.out.println(a.f(3));
16  } }
```

Test 6 java Otto

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

A

Test 7: java Main a b c

```
01 public class Main {
02     int MAIN=10;
03     Main(){
04         MAIN++;
05     }
06     void main(String args){
07         MAIN--;
08     }
09     public static void main(String[] args) {
10         Main maiN=new Main();
11         maiN.MAIN++;
12         maiN.main(args[0]);
13        System.out.print(maiN.MAIN);
14     }
15 }
```

Test 8 java Sette

```
01 import java.util.*;
02 public class Sette {
03     Sette(){
04         Collection<String> a = new ArrayList<String>();
05         Collection<String> b = new HashSet<String>();
06         for (int k=0;k<10;k++) {
07             String s="A"+(k%4);
08             a.add(s);
09             b.add(s);
10         }
11         int count=0;
12         Iterator<String>i=a.iterator();
13         while (i.hasNext()) {
14             String s=i.next();
15             count++;
16         }
17         i=b.iterator();
18         while (i.hasNext()) {
19             String s=i.next();
20             count++;
21         }
22         System.out.println(count);
23     }
24     public static void main(String[] a) {
25          new Sette();
26 }}
```

A

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

9.1	Su qualunque oggetto si può chiamare il metodo clone per ottenerne una copia.
9.2	Il metodo equals applicato a due oggetti della stessa classe restituisce true se i due oggetti hanno lo stesso stato.
9.3	Da un punto di vista di efficienza di esecuzione lo static binding è meglio del dynamic binding.
9.4	A differenza del C, Java usa solo la heap e non lo stack.
9.5	Un oggetto ed un suo clone sono identici.
9.6	Ogni metodo di una classe java può leggere il valore di una qualunque variabile di istanza della classe stessa.
9.7	Esistono metodi che possono essere eseguiti anche senza creare istanze di una classe.
9.8	L'ereditarietà multipla è ammessa per le interfacce.