ESAME 12/06/11

B

Test 1 java B

```
01 public class B 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         ((B)copia).k++;
11         return copia;
12     }
13     public boolean equals(Object x) {
14         if (! (x instanceof B)) return false;
15         return k==((B)x).k;
16     }
17     public static void main(String[] args) {
18         B b= new B();
19         B c=(B)b.clone();
20         B d=new B();
21         if (b.equals(c)) System.out.print("Z");
22         if (c.equals(d)) System.out.print("Y");
23         if (d.equals(b)) System.out.print("X");
24 }}
```

Test 2 java B

```
01 class A {
02   A(int x) {System.out.print("A");}
03   A() {System.out.print("S");}
04   public void finalize() {System.out.print("U");}
05 }
06 public class B extends A {
07   B(int x) {System.out.print("T");}
08   B() {System.out.print("H");}
09   public void finalize() {System.out.print("P");}
10   public static void main(String args[]) {
11     A a=new B(3);
12     a = null;
13     System.gc();
14     System.runFinalization();
15   }
16 }
```

Test 3 java B

```
00 class B {
01     static 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(b2)) System.out.print("3"); else
   System.out.print("1");
10         if (b1.s==b3.s) System.out.print("3"); else
   System.out.print("1");
11 }   }
```

B

Test 4 java B

```
00 public class B {
01     String a[10];
02     void initialize(){
03         for (int k=9;k>=0;--k) a[k]=""+k;
04     }
05     void stampa(int k){
06         System.out.println(a[k]);
07     }
08     B() {initialize(); stampa(0); }
09     public static void main(String a[]){
10         new B();
11     }
12     public static void main(String a){
13         new B();
14     }
15 }
```

Test 5 java B

```
00 public class B {
01     static String k="pluto";
02     public static void main(String a[]){
03         new B();
04         initialize("pippo");
05         System.out.println(k);
06     }
07     void main(){
08         new B();
09     }
10     void initialize(String s){k=s;}
11 }
```

Test 6 java B

```
01 class C{
02     void f() {
03       System.out.println("R");
04     }
05 }
06 public class B extends C{
07    public void f() {
08      System.out.println("P");
09    }
10   public static void main(String[] args) {
11     C quattro1 = new B();
12     quattro1.f();
13   }
13   public static void main(String args) {
13     C αλΦα = new C();
13     αλΦα.f();
13   }}
```

B

Test 7 java B

```
01  class B {
02        public static void main(String args[]) {
03              int i,j,k,l=8;
04              k = l++;
05              j = ++k;
06              i = j++;
07              System.out.println(i);
08        }
09  }
```

Test 8 java B

```
01  public class B {
02    void f(int k) {
03      System.out.print(k*3);
04    }
05    public static void main (String args[]){
06      Object z = new A();
07      if (z instanceof B) ((B) z).f(4);
08      if (z instanceof A) ((A) z).f(2);
09    }
10  }
11  class A extends B{
12    void f(int k) {
13      System.out.print(k*2);
14    }
15  }
```

Test 9 – scrivere nel campo per l'output del test la sequenza risultante indicando T per le affermazioni vere e F per quelle false. Tutte le affermazioni, ove non altrimenti specificato, riguardano Java.

9.1	In Java non esistono le variabili globali
9.2	Una classe figlia può fare overloading di un metodo final della classe padre
9.3	Una classe figlia può fare l'overriding di un metodo final della classe padre
9.4	Se una classe è astratta è permesso usarla per effettuare ereditarietà multipla
9.5	Due oggetti per cui equals è vero possono avere variabili pubbliche con valori differenti
9.6	Due oggetti per cui equals è vero sono sempre identici
9.7	Se una Collection è specializzata tramite una generic gli oggetti estratti dal relativo iteratore non richiedono un cast.
9.8	Java usa solo la heap e non lo stack