Test 1 

```
00  public class Sei {
01      char f() { return '6'; }
02      public static void main(String e[]) {
03          Sei a = new Sei();
04          Sei b = new Sette();
05          Sette c = new Sette();
06          System.out.print(a.f() + " " + b.f() + " " + c.f() + "
    ");
07          char ch[] = {'A', 'B', 'A', 'B', 'A', 'B'};
08          int i1 = 0, i2 = 2, i3 = 4;
09          if (a.equals(b)) i1++;
10          if (b.equals(a)) i2++;
11          if (c.equals(b)) i3++;
12          System.out.println(ch[i1] + " " + ch[i2] + " " +
    ch[i3]);
13  }   }
14  class Sette extends Sei {
15      char f() { return '7'; }
16      public boolean equals(Object a) {
17          return (a instanceof Sei);
18      }
19      public int hashCode() { return 3; }
20  }
```

Test 2

```
00  public class Uno {
01      static Collection c=new HashSet();
02      public static void main(String a[]) {
03          Collection c=new LinkedList();
04          Uno u=new Uno();
05          c.add(u); c.add(u);
06          u.f();
07          System.gc();System.runFinalization();
08          System.out.println(c.size());
09      }
10      void f() {
11          A a=new A("K");
12          A b=new A("L");
13          c.add(b);
14      }
15      class A {
16          String s="";
17          A(String s) {this.s=s; System.out.print(this);}
18          public String toString(){return s;}
19          public void finalize(){System.out.print(this);}
20  }   }
```

Test 3

```
01  public class Due {
02      static Collection<Due> s=new HashSet<Due>();
03      int k,j;
04      Due(int k, int j) {this.k=k;   this.j=j;}
05      public boolean equals(Object d){
06          return k-j==((Due)d).j-((Due)d).k;
07      }
08      public int hashCode(){return 1;}
09      public static void main(String[] m){
10          s.add(new Due(1,2)); s.add(new Due(0,1));
11          s.add(new Due(2,1)); s.add(new Due(1,0));
12          System.out.print(s.size());
13          for (Due x:s){System.out.print(x.k+" "+x.j);}
14      }
15      public static void main(String m){
16          s.add(new Due(1,0));
17          System.out.print(s.size());
18  }   }
```

Test 4

```
01  public class Due {
02      static Collection<Due> s=new HashSet<Due>();
03      static int k,j;
04      Due(int k, int j) {this.k=k;   this.j=j;}
05      public boolean equals(Object d){
06          return k-j==((Due)d).j-((Due)d).k;
07      }
08      public int hashCode(){return 1;}
09      public static void main(String[] m){
10          s.add(new Due(1,2)); s.add(new Due(0,1));
11          s.add(new Due(2,1)); s.add(new Due(1,0));
12          System.out.print(s.size());
13          for (Due x:s){System.out.print(x.k+" "+x.j);}
14  }   }
```

Test 5

```
01  public class Due {
02      Collection<Due> s=new HashSet<Due>();
03      static int k,j;
04      Due(int k, int j) {this.k=k;   this.j=j;}
05      public boolean equals(Object d){
06          return k-j==((Due)d).j-((Due)d).k;
07      }
08      public int hashCode(){return 1;}
09      public static void main(String[] m){
10          s.add(new Due(1,2)); s.add(new Due(0,1));
11          s.add(new Due(2,1)); s.add(new Due(1,0));
12          System.out.print(s.size());
13          for (Due x:s){System.out.print(x.k+" "+x.j);}
14  }   }
```

Test 6

```
00  public class Tre {
01      class A {
02          public A(int k) {System.out.print(k);}
03          public void finalize() {System.out.print("A");}
04      }
05      class B extends A {
06          public B(int k) {System.out.print(k);}
07          public void finalize() {System.out.print("A");}
08      }
09      public static void main (String z[]){
10         new Tre();
11      }
12      Tre(){
13          A a=new B(3);
14          B b=(B)a;
15          a=null;
16          b=new B(3);
17          System.gc(); System.runFinalization();
18  }    }
```

Test 7 

```
01  #include <iostream>
02  using namespace std;
03  int x[] = {-2, -1, 0, 1, 2};
04  void f(int* x, int y[]) {
05      x[*y] = -y[*x];
06  }
07  int main(int argc, char** argv) {
08      int * p = x + 1;
09      f(p, p);
10      for (int * s = x; s < x + 5; s++) {
11          cout << *s;
12      }
13      return 0;
14  }
```

Test 8 

```
01  public class F{
02      int x=2;
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      }}
```

A (giuste)

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	int a[] è un oggetto.
9.2	Il metodo finalize() chiama automaticamente il corrispondente metodo della superclasse
9.3	Il costruttore chiama automaticamente il costruttore della superclasse con gli stessi parametri. Se nella superclasse non è disponibile un costruttore con la stessa firma, viene chiamato il costruttore vuoto.
9.4	L'esistenza in una classe di un metodo f(int x) e di uno f(String s) è un esempio di overloading
9.5	L'esistenza in una classe di un metodo f(int x), e in una sua superclass di un metodo f(String s) è un esempio di overriding
9.6	Una classe astratta può implementare al massimo una interfaccia
9.7	Se una classe è astratta è permesso usarla per effettuare ereditarietà multipla
9.8	In un programma ci possono essere più classi con lo stesso nome