

B

Test 1

00	public class C {
01	int s=4;
02	void f() {System.out.print("A"++s);}
03	public static void main(String[]a){
04	C y=new C();
05	C x=new C() {
06	void f() {System.out.print("B"+(--s));}
07	};
08	x.f();
09	y.f();
10	}
11	public static void main(String a){
12	C y=new C();
13	}
14	}

Test 2

01	using namespace std;
02	#include <cstdlib>
03	#include <iostream>
04	void f(int x[4],int value){ x[0]+=value; }
05	void g(int x,int value){ x+=value; }
06	int main(int argc, char** argv) {
07	int z=5;
08	int a[]={1,2,3,4,5,6,7,8,9};
09	g(z,1); <i>NICHE</i>
10	f(&z,3); <i>z=8</i>
11	f(a+3,3); <i>z=</i>
12	g(a[3],4); <i>NICHE</i>
13	cout<<z<<" "<<a[3]; <i>8 8 7</i>
14	return 0;
15	}

Test 3

01	class A1 {int x=2; }
02	class B1 extends A1 {int k=1;}
03	public class E1 {
04	public static void main(String[] args) {
05	A1 a1=new B1();
06	A1 a2=(A1)(new B1());
07	B1 b1=new A1();
08	System.out.println(a1.x+a2.x+b1.x);
09	}
10	}

B

Test 4

00	public class E6 {
01	static Collection ll = new LinkedList();
02	int x=1;
03	E6(int x){
04	ll.add(this);
05	ll.add(new E6A());
06	}
07	public static void main(String z[]) {
08	new E6(3);
09	Iterator iter = ll.iterator();
10	while (iter.hasNext()) {
11	((E6)(iter.next())).f();
12	}}
13	public void f() { System.out.print(x); }
14	public static void main(String z) {
15	new E6A();
16	System.out.print(ll.size());
17	}}
18	class E6A extends E6 {
19	E6A(){x++;}
20	public void f() {
21	x++; super.f(); System.out.print(2);
22	}}

Test 5

00	public class D {
01	static int x=3; 4807
02	public static void main(String[] args) {
03	D a5=new D(); a5.f();
04	a5=new D(); a5.f();
05	System.gc(); System.runFinalization();
06	}
07	void f() {Pippo a=new Pippo2();
08	}
09	public void finalize() { System.out.print("X"); }
10	class Pippo {
11	int k;
12	Pippo() {k=++x;}
13	public void finalize() { System.out.print(k); }
14	}
15	class Pippo2 extends Pippo {
16	Pippo2() {k=x++;}
17	}
18	}

B

Test 6

01	public class E2 {
02	static HashSet hs=new HashSet();
03	public int hashCode() {return 0;}
04	public boolean equals(Object x) {
	return (x.getClass().equals(this.getClass()));}
05	public static void main(String s[]){
06	f(new E2()); f(new E2()); f(new A2());
07	f(new A2()); f(new A3()); f(new A3());
08	}
09	static void f(E2 x) {int v=0;
	if (hs.add(x)) v=1; System.out.print(v);
	}
10	}
11	class A2 extends E2 {
12	public boolean equals(Object x) {return x instanceof A2;}
13	}
14	class A3 extends A2 {}

Test 7

01	public class E3 {
02	static int counter=0; 173
03	private int value=0;
04	E3(){value++;counter;}
05	public String toString(){
06	return this.getClass().getName()+value+" ";
07	public void finalize(){System.out.print("F"+value);}
08	}
09	class G extends E3{
10	public static void main(String d[]){
11	LinkedList<E3> x=new LinkedList<E3>();
12	E3 a1=new G();
13	G a2=new G();
14	E3 a3=new E3(); 5
15	x.add(a1);x.add(a3);
16	a1=null; a2=null; a3=null;
17	Iterator<E3> it=x.iterator();
18	while (it.hasNext()){System.out.print(it.next());}
19	System.gc();System.runFinalization();
20	}}

Test 8

00	public class E4 {
01	int x=2; 34
02	E4(int x) {
03	f(x); f();
04	System.out.println(x);
05	}
06	void f() { x++; System.out.print(x);}
07	void f(int x) { this.x++; x--;System.out.print(x);}
08	public static void main(String arg[]) {
09	int x=2;
10	new E4(2);
11	}}

B

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	Se <code>a.hashCode()!=b.hashCode</code> , <code>a.equals(b)</code> deve essere falso
9.2	Nel main si può leggere una qualunque variabile di istanza della classe in cui è contenuto.
9.3	Ereditarietà multipla è permessa con le interfacce e le classi astratte.
9.4	E' corretto scrivere <code>Integer k=3;</code>
9.5	L'esistenza di un metodo <code>f(int x)</code> in una classe e di uno <code>f(String s)</code> in una sua sottoclasse è un esempio di overloading
9.6	In una classe ci può essere un solo metodo main
9.7	Il metodo <code>finalize()</code> chiama automaticamente il corrispondente metodo della superclasse
9.8	Il costruttore chiama automaticamente il costruttore della superclasse con gli stessi parametri. Se nella superclasse non è disponibile un costruttore con parametri dello stesso tipo, viene chiamato il costruttore vuoto.