1.

**package** oops;

**public** **class** Singleton {

**private** **static** Singleton *single\_instance* = **null**;

**public** String s;

**private** Singleton()

{

s = "Hello I am a string part of Singleton class";

}

**public** **static** Singleton getInstance()

{

**if** (*single\_instance* == **null**)

*single\_instance* = **new** Singleton();

**return** *single\_instance*;

}

}

**package** oops;

**public** **class** main {

**public** **static** **void** main(String args[])

{

Singleton x = Singleton.getInstance();

Singleton y = Singleton.getInstance();

Singleton z = Singleton.getInstance();

System.out.println("Hashcode of x is " + x.hashCode());

System.out.println("Hashcode of y is " + y.hashCode());

System.out.println("Hashcode of z is " + z.hashCode());

**if** (x == y && y == z)

{

System.out.println("Three objects point to the same memory

location on the heap i.e, to the same object");

}

**else** {

System.out.println("Three objects DO NOT point to the same memory

location on the heap");

}

}

}

**2.package** pgm1;

**import** java.util.Scanner;

**public** **class** emp {

**public** **static** **void** main(String[] args)

{

manager m=**new** manager();

m.emp();

m.incentive();

labour l=**new** labour();

l.emp();

l.OT();

}

}

**package** pgm1;

**public** **class** employee {

**void** emp()

{

System.out.println("employee salary");

}

}

**package** pgm1;

**import** java.util.Scanner;

**public** **class** manager **extends** employee {

**void** incentive()

{

Scanner scan=**new** Scanner(System.in);

System.out.println("incentive:");

**int** ince;

ince=scan.nextInt();

ince=ince+10000;

System.out.println("salary of the manager:"+ince);

}

}

**package** pgm1;

**import** java.util.Scanner;

**public** **class** labour **extends** employee {

**void** OT()

{

Scanner scan=**new** Scanner(System.in);

System.out.println("OT:");

**int** OT;

OT=scan.nextInt();

OT=OT+10000;

System.out.println("salary of the labour:"+OT);

}}

Output:

employee salary

incentive:

1000

salary of the manager:11000

employee salary

OT:

200

salary of the labour:1020

3. **public** **class** bank {

**public** **void** display()

{

System.***out***.println("account");

}

}

**public** **class** saving **extends** bank{

**public** **void** cash()

{ **int** number=20000;

System.***out***.println("saving holder");

System.***out***.println("Fixed Deposit"+number);

}

}

**public** **class** current **extends** bank {

**public** **void** cash()

{ **int** number=230000;

System.***out***.println("current holder");

System.***out***.println("Credit Cash"+number);

}

}

**public** **class** hdfc {

**public** **static** **void** main(String[] args)

{

saving s=**new** saving();

s.cash();

current c=**new** current();

c.cash();

}

}

Output:

saving account holder

Fixed deposit:10000

Current account holder

credit cash:5000

4. Test the following principle regarding abstraction

i. if any class has any of its method abstract then you must declare entire class abstract

**package** abstarct;

**public** **class** demo **extends** call {

**public** **void** dispaly2()

{

System.out.println("HI");

}

**public** **static** **void** main(String[] args)

{

demo d= **new** demo();

d.dispaly();

}

}

**package** abstarct;

**abstract** **class** call {

**void** display()

{

System.out.println("hello");

}

}

Output: abstract class have abstract method and methods do not require implementation for declaration

ii.

**package** abstarct;

**public** **class** demo **extends** call {

**public** **void** dispaly2()

{

System.out.println("HI");

}

**public** **static** **void** main(String[] args)

{

demo d= **new** demo();

d.dispaly2();

}

}

**package** abstarct;

**abstract** **class** call {

**void** display()

{

System.out.println("hello");

}

}

Output: HI

iii. Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Implicit super constructor call() is not visible for default constructor. Must define an explicit constructor

Iv . Abstarct class cannot be final because final is not a access specifier

v. yes

5. **public** **abstract** **class** shape

{

**abstract** **void** draw();

}

**public** **class** rectangle **extends** shape

{

@Override

**public** **void** draw() {

System.***out***.println("Rectangle"); // **TODO** Auto-generated method stub

}

}

**public** **class** line **extends** shape

{

**public** **void** draw()

{

System.***out***.println("Line"); // **TODO** Auto-generated method stub

}

}

**public** **class** cube **extends** shape

{

**public** **void** draw(){

System.***out***.println("cube");

}

}

**public** **class** poly {

**public** **static** **void** main(String[] args) {

shape s=**new** rectangle();

shape s1=**new** line();

shape s2=**new** cube();

s.draw();

s1.draw();

s2.draw();

}

}