Practical 7

Method Overloading:-Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.
Method overloading is also known as ***Compile-time Polymorphism***, *Static Polymorphism, or* ***Early binding*** in Java. In Method overloading compared to parent argument, child argument will get the highest priority.

Code->

```
public class ComplexNumber
{
int real, image;

public ComplexNumber(int r, int i)
{ this.real = r;
this.image = i;
}

public void showC()
{ System.out.print(this.real + " +i" + this.image);
}

public static ComplexNumber add(ComplexNumber n1, ComplexNumber n2) { ComplexNumber res = new ComplexNumber(0, 0);

res.real = n1.real + n2.real; res.image = n1.image + n2.image;
return res;
}

public static void main(String arg[]){
System.out.println("Aditi Sharma");
{

ComplexNumber c1 = new ComplexNumber(4, 5);
ComplexNumber c2 = new ComplexNumber(10, 5);
System.out.print("first Complex number: ");
c1.showC();

System.out.print("\nSecond Complex number: ");
c2.showC();
ComplexNumber res = add(c1, c2);
System.out.print("\nAddition is :");
res.showC();
}
}
}
}
```

Output:-

Output

```
java -cp /tmp/5qPcoTm2PU ComplexNumber
Aditi Sharma
first Complex number: 4 +i5
Second Complex number: 10 +i5
Addition is :14 +i10
```

Practical No:- 8

Constructor Overloading by passing objects as arguments

Code->

```
 import java.util.*;
class area {
double length, breadth;

area()
{ length = 0;
breadth = 0;
}

area(int l, int b)
{ length = l; breadth = b;
}

area(int l)
{
length = breadth = l;
}

double area()
{
return length * breadth;

}
}
public class Rectangle
{

public static void main(String args[])
{System.out.println("Aditi Sharma");
{ area r1 = new area();
area r2 = new area(5);
area r3 = new area(2, 6);
System.out.println("area of rectangle 1 is: " + r1.area());
System.out.println("area of rectangle 2 is: " + r2.area());
System.out.println("area of rectangle 3 is: " + r3.area());
}
}
}
```

Output:-

```
Aditi Sharma
area of rectangle 1 is: 0.0
area of rectangle 2 is: 25.0
area of rectangle 3 is: 12.0
```

Practical No:- 9

Various access control and usage of static, final and finalize ( )

code->

```
public class MyClass
{ final int age = 18;

void showAge() {
   age = 55;
}

public static void main(String[] args)
{System.out.println("Aditi Sharma");
{ MyClass student = new MyClass();
student.showAge();
}
}
}
```

Output:->

```
/MyClass.java:5: error: cannot assign a value to final variable age
    age = 55;
    ^
1 error
```

finalize:

Code->

```
public class finalize {
public static void main(String[] args) {
finalize str2 = new finalize();
str2 = null;

System.gc();
System.out.println("output of main method");
}

protected void finalize() {
System.out.println("output of finalize method");
}
}
```

Output->

```
output of main method
output of finalize method
```

Static:

Code->

```
public class static1 {
// static method
static void show() {
System.out.println("Calling method without creating any object of class");
}

public static void main(String[] args)
{System.out.println("Aditi Sharma");
{ show();
}
}
}
```

Output:-

```
Aditi Sharma
Calling method without creating any object of class
```

Practical No :- 10

Command line arguments

Code->

```
public class Main {
public static void main(String args[])
{System.out.println("Aditi Sharma");
{ for (int i = 0; i < args.length; i++)
{
System.out.println("args[" + i + "]: " + args[i]);
}
}
}
}
```

Output->

```
Aditi Sharma
Aditi
Vani
Tanu
Anku
```

Practical No :- 11

Inheritance in Java

Code->

```
import java.io.*;
import java.lang.*;
import java.util.*;

class one {
   public void print_geek()
   {
      System.out.println("Geeks");
   }
}

class two extends one {
   public void print_for()
{ System.out.println("for"); }
}
// Driver class
public class Main {
   public static void main(String[] args)
   {System.out.println("Aditi Sharma");
   {
      two g = new two();
      g.print_geek();
      g.print_for();
      g.print_geek();
   }
}
}
```

Output->

```
Aditi Sharma
Geeks
for
Geeks
```

Practical No :- 12

Method Overriding

Code->

```
class Parent { void fun() {
System.out.println("parent's fun");
}
}

class Child extends Parent { @Override
void fun()
{ System.out.println("child's fun");
}
}

class GrandChild extends Child { @Override
void fun()
{ System.out.println("Grandchild's fun");
}
}

public class overriding {
public static void main(String args[])
{System.out.println("Aditi Sharma");
{ Parent p = new Child();
Parent ch = new GrandChild();
p.fun();
ch.fun();

}
}
}
```

Output:-

```
Aditi Sharma
child's fun
Grandchild's fun
```

Practical No :-13

Abstract classes

Code->

```
abstract class parent {
abstract void write();
}

class child extends parent { @Override
void write()
{ System.out.println("Writing.          ");
}

}
public class Abstract{

public static void main(String args[])
{System.out.println("Aditi Sharma");
{ child a = new child();
a.write();
}
}
}
```

Output:-

```
Aditi Sharma
Writing.
```

Practical No :-14

Nested class

Code->

```
class outer {
int x = 9;

class nested { int y = 4;
}
}

public class nestedClass {
public static void main(String[] args)
{System.out.println("Aditi Sharma");
{ outer otr = new outer();
outer.nested nstd = otr.new nested();

System.out.println("Variable of outer class : " + otr.x);
System.out.println("Variable of nested class : " + nstd.y);
}
}
}
```

Output:-

```
Aditi Sharma
Variable of outer class : 9
Variable of nested class : 4
```