**Assignment**

Creating method with void and parameter?

**public** **class** multiplication

{

**public** **static** **void** addition(**int** A , **int** B)

{

**Int** X=A+B;

System.***out***.println("Addition of A & B:"+X);

}

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

{

*Addition* (1,4);

}

}

Creating method with return data type and parameter?

Calling static method

Using static property

**public** **class** Substraction

{

**public** **static** **int** substaction(**int** i , **int** j)

{

**int** x = i-j;

**return** c;

}

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

{

**int** a = *substraction*(1,3);

System.***out***.println(a);

}

}

O/P: -2

Creating object?  
Calling method with return and storing the return data

**MethodExecution**

**package** work;

**public** **class** program

{

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

{

MethodDef MD = **new** MethodDef();

MD. substraction ();

MD.addition();

}

}

**MethodDef**

**package** work;

**public** **class** MethodDef

{

**int** i = 7, j = 5;

**public** **void** addition()

{

**int** c = i + j;

System.***out***.println(c);

}

**public** **void** substraction()

{

**int** c = i - j;

System.***out***.println(c);

}

}

Calling method with return and parameter?

MethodExecution

**package** work;

**public** **class** MethodDef

{

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

{

MethodDef MD = **new** MethodDef();

**int** a=MD.addition(7,3);

**int** b=MD.substraction(7,3);

System.***out***.println(a);

System.***out***.println(b);

}

}

MethodDef

**package** work;

**public** **class** MethodDef

{

**public** **int** addition(**int** i,**int** j)

{

**int** c = i + j;

**return** c;

}

**public** **int** substraction(**int** i, **int** j)

{

**int** c = i - j;

**return** c;

}

}

Create classes under multiple packages?

Calling classes under different packages

Method Execution

**package** work;

**import** work1.PackageMethod;

**public** **class** program

{

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

{

PackageMethod pm = **new** PackageMethod();

**int** a = pm.addition(7, 3);

**int** b = pm.substraction(7, 3);

System.***out***.println(a);

System.***out***.println(b);

}

}

PackageMethod

**package** work1;

**public** **class** PackageMethod

{

**public** **int** addition(**int** i, **int** j)

{

**int** c = i + j;

**return** c;

}

**public** **int** substraction(**int** i, **int** j)

{

**int** c = i - j;

**return** c;

}

}

Write code for interface and create class to implement that interface

**public** **interface** InterfaceMethod

{

**public** **void** add();

}

**public** **class** InterfaceMethodDeclare **implements** InterfaceMethod {

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

{

InterfaceMethodDeclare imd = **new** InterfaceMethodDeclare ();

imd.add();

}

**public** **void** add()

{

**int** i = 5, j = 2;

**int** c = i+j;

System.***out***.println(c);

}

}

what is **Final keyword**?

In the Java programming language, the final keyword is used in several different contexts to define an entity that can only be assigned once.. Once a final factor has been relegated, it generally contains the same value.Implement method overriding

**public** **class** Shape

{

{

double dim1,dim2;

public shape(double d1, double d2)

{

dim1=d1;

dim2=d2;

}

abstract public double area();

{

//return 0.0;

}

}

**public** **class** Rectangle extends Shape

{

public Rectangle (double d1,double d2)

{

super(d1,d2)

}

public double Area()

{

return dim1\*dim2;

}

public static void main(string args[])

{

Rectangle re=New Rectangle (10,20);

system.out,println(re.Area());

}

}

Write code to handle exceptions with try/catch/finally?

**public** **class** Exceptions

{

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

{

**try**

{

**int** a = a , b = 0;

**int** c = a/b;

System.***out***.println(c);

}

**catch**(ArithmeticException e)

{

System.***out***.println("exception exists, check the values");

}

**finally**

{

System.***out***.println("This is finally block");

}

}

}

Write code for creating abstract class

Abstract Class

**abstract public** **class** shape

{

double dim1, dim2;

public shape (double d1, double d2)

{

dim1=d1;

dim2=d2;

}

abstract public double area();

{

//return 0.0;

}

}

Implement method overloading

**public** **class** Example

{

**public** int abs(**int** a)

{

System.***out***.println("Integer version");

**return** a<0?a\*-1:a;

}

**public** float abs(**float** a)

{

System.***out***.println("Float version");

**return** a<0?a\*-1:a;

}

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

{

Example e=**new** Example ();

System.***out***.println(e.abs(-10));

System.***out***.println(e.abs(-10f));

}

}