**create objects in Java**

As you all know, in Java, a class provides the blueprint for objects, you create an object from a class.

Basically class act as a container where we collect attribute and behaviour of objects. By using instance or object of a class we can able to access member of class.

Using new Keyword : Using new keyword is the most basic way to create an object. This is the most common way to create an object in java. Almost 99% of objects are created in this way. By using this method we can call any constructor we want to call (no argument or parameterized constructors).

// Java program to illustrate creation of Object

// using new keyword

public class A

{

String name = "king";

public static void main(String[] args)

{

// Here we are creating Object of

// A using new keyword

A obj = new A();

System.out.println(obj.name);

}

}

Output

King

**We can able to create n number of object or instance for a class. For every instance of a class separate copy of instance member will create.**

class A

{

void add(int a,int b)

{

int s;

s=a+b;

System.out.println(s);

}

public static void main(String args[])

{

A x=new A();

A y=new A();

x.add(10,12);

y.add(3,4);

}

}

Output:

22

7

In the above example , we can find out two objects for class A and for every object separate copy of add method created.

**We can able to invoke one member of a class n number of times as per our requirement by using same object reference.**

class A

{

void add(int a,int b)

{

int s;

s=a+b;

System.out.println(s);

}

public static void main(String args[])

{

A x=new A();

x.add(10,2);

x.add(3,5);

}

}

Output

12

8

In the above example we invoke add method twice by using same reference of class A.

**METHOD OVERLOADING**

If a class has multiple methods having same name but different signatures, it is known as Method Overloading. Signature of a method indicates no of parameter as well as type of parameter.

**In the concept of method overloading at compile time by knowing parameter control recognize which method will invoke, that’s why method overloading is the example of compile time polymorphism.**

If we have to perform only one operation, having same name of the methods increases the readability of the program.

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 both. Overloading is related to compile-time (or static) polymorphism.

public class Sum {

// Overloaded sum(). This sum takes two int parameters

public int sum(int x, int y)

{

return (x + y);

}

// Overloaded sum(). This sum takes three int parameters

public int sum(int x, int y, int z)

{

return (x + y + z);

}

// Overloaded sum(). This sum takes two double parameters

public double sum(double x, double y)

{

return (x + y);

}

public static void main(String args[])

{

Sum s = new Sum();

System.out.println(s.sum(10, 20));

System.out.println(s.sum(10, 20, 30));

System.out.println(s.sum(10.20, 20.50));

}

}

Output

30

60

30.70