02

INTRODUCTION TO OOP AND BASIC JAVA PROGRAMMING.

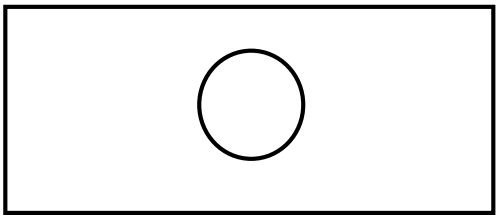
OBJECT ORIENTED PROGRAMMING LAB

SESSION - 02

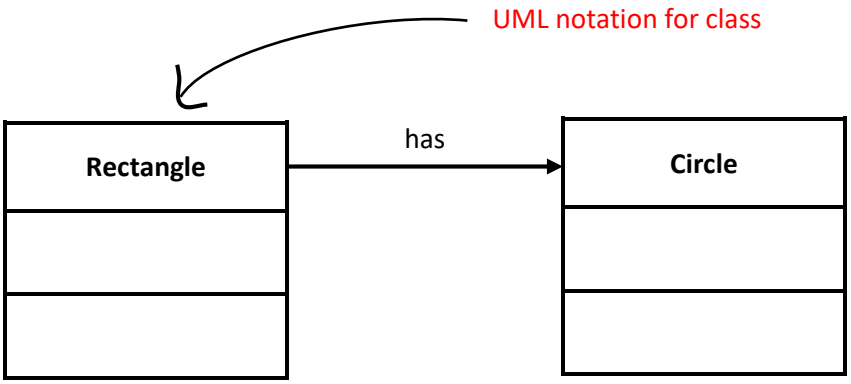
UML (Unified Modeling Language)

- UML (Unified Modeling Language) is a standard notation for the modeling of real-world objects as a first step in developing an object-oriented design methodology.

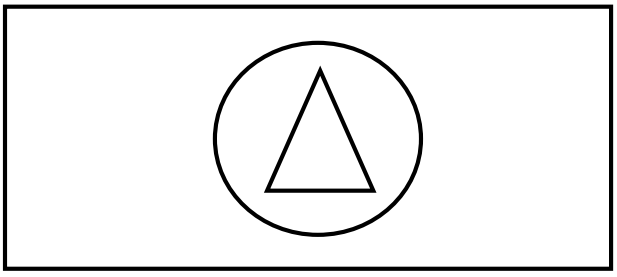
#01. Draw the UML model for the following figure.



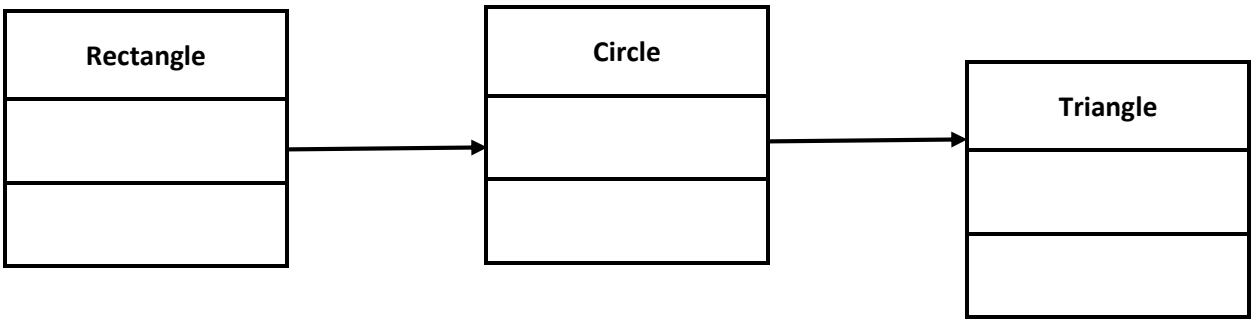
Sol^{n}:



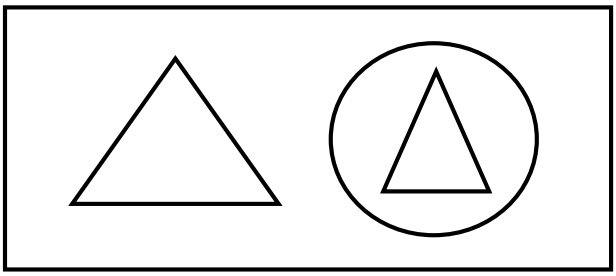
#02. Draw the UML model for the following figure.



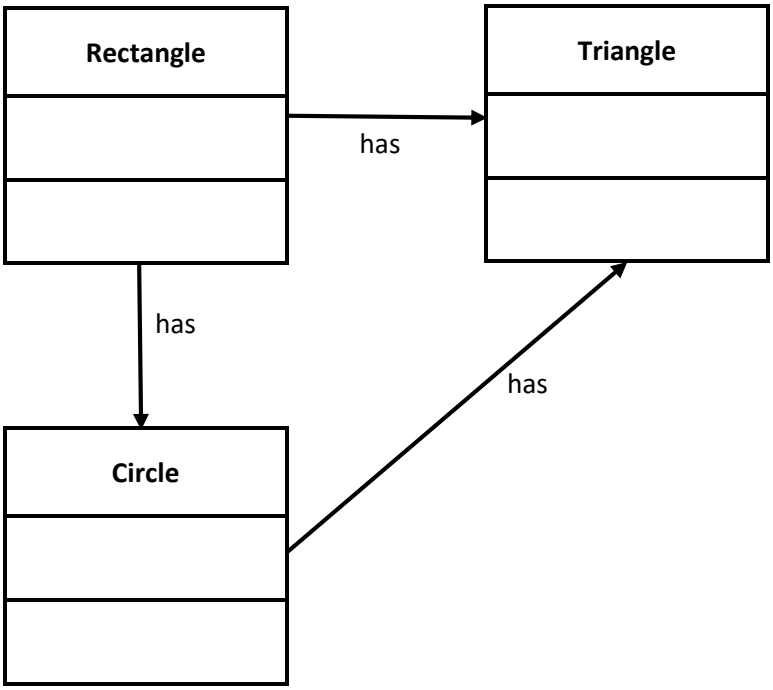
Sol^{n}:



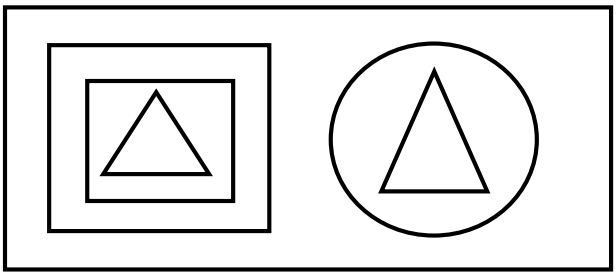
#03. Draw the UML model for the following figure.



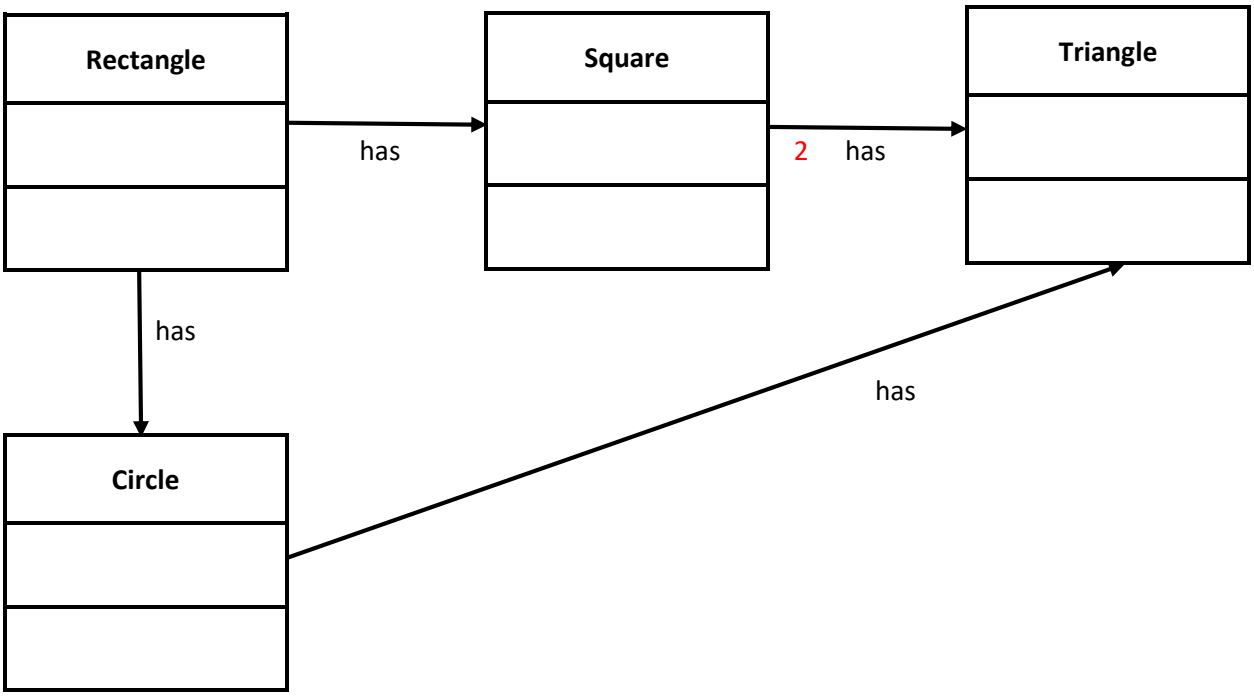
Sol^{n}:



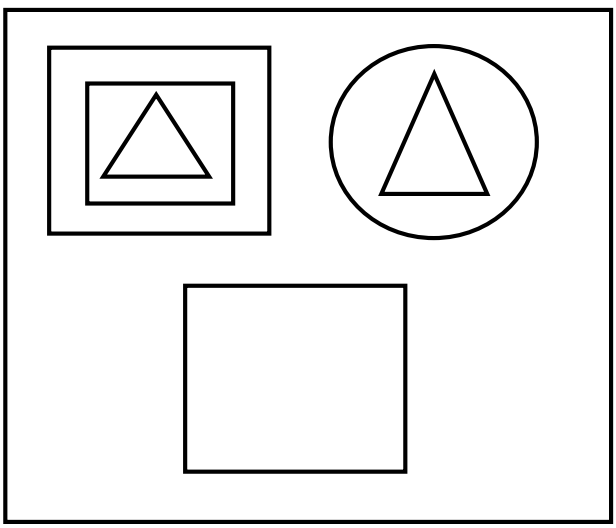
#04. Draw the UML model for the following figure.



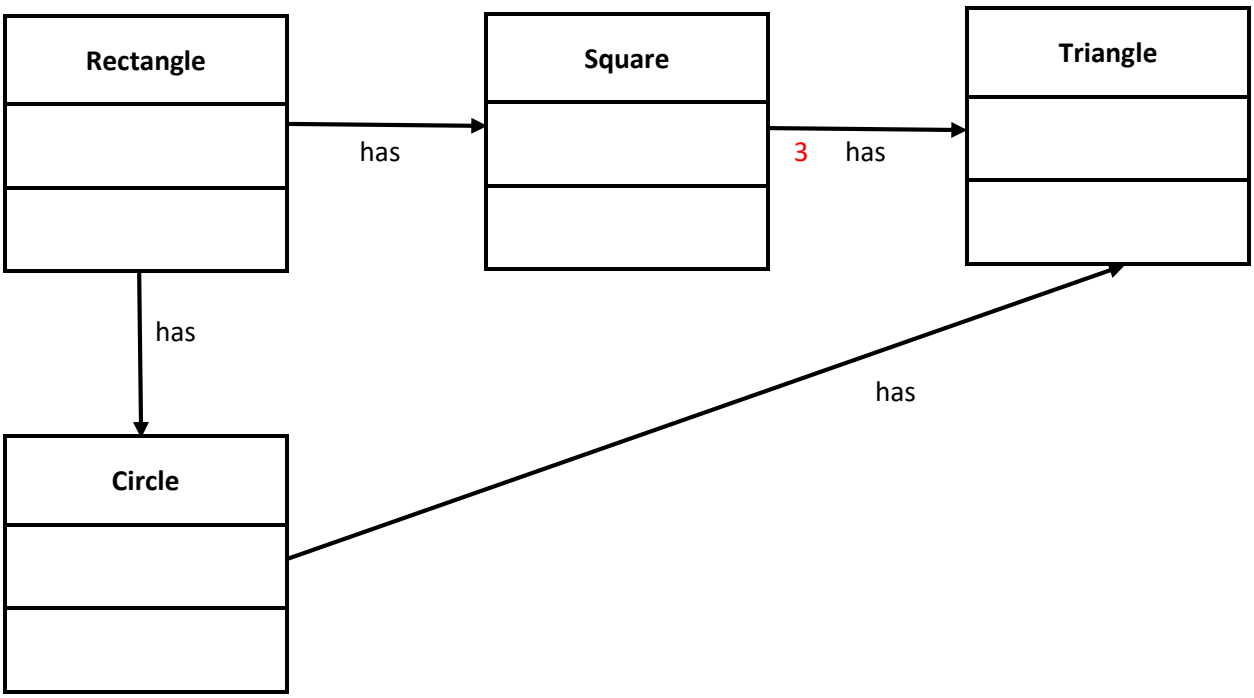
Sol^{n}:



#05. Draw the UML model for the following figure.



Sol^{n}:



Variable and its Type:

Data type:

Primitive type:

these are primitive data types

- boolean
- character
- byte
- short
- integer
- long
- float
- double

Each variable in Java has a specific type, which determines the size and layout of the variable's memory. Following are valid examples of variable declaration and initialization in Java –

```
int a, b, c;         // Declares three ints a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22;         // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a';        // the char variable a iis initialized with value 'a'
```

There are three kinds of variables in Java –

- Local variables
- Instance variables
- Class/Static variables

Local Variables:

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

```
public class Test {
  public void Age() {
    int age = 0;
    age = age + 7;
    System.out.println("Puppy age is : " + age);
  }
  public static void main(String args[]) {
    Test test = new Test();
    test.Age();
  }
}
```

Output is-

```
Puppy age is: 7
```

Instance Variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Reference Variable:

- **Reference** variables are used to refer to an object. They are declared with a specific type which cannot be changed.
- **Reference variables** can be declared as static **variables**, instance**variables**, method parameters, or local **variables**.

```
public class Employee {
     public String name;   // this instance variable is visible for any child class.
     private double salary;   // salary  variable is visible in Employee class only.
     public Employee (String empName) {
        name = empName;   // The name variable is assigned in the constructor.
    }
     public void setSalary(double empSal) {
        salary = empSal;   // The salary variable is assigned a value.
     }
     public void printEmp() {
         // This method prints the employee details.
         System.out.println("Name  : " + name );
         System.out.println("Salary :" + salary);
    }
    public static void main(String args[]) {
        Employee empOne = new Employee("Ransika");  //empOne is reference variable
        empOne.setSalary(1000);
        empOne.printEmp();
    }
}
```

Output is-

```
Name  : Ransika
Salary :1000.0
```

Class/Static Variables:

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned

during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.

- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

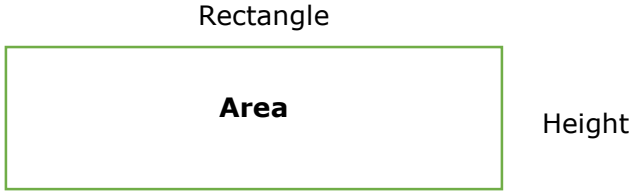
```
import java.io.*;
public class Employee {
   private static double salary;   // salary  variable is a private static variable
   public static final String DEPARTMENT = "Development ";   // DEPARTMENT is a constant
   public static void main(String args[]) {
       salary = 1000;
       System.out.println(DEPARTMENT + "average salary:" + salary);
   }
}
```

Access Modifier:

There are 4 types of java access modifiers:

1. Private: accessible only within class.
2. Default: accessible only within package.
3. Protected: accessible within package and outside the package but through inheritance only.
4. Public: accessible everywhere.

Write a code in Java:




```
public class Rectangle {
        private int height;
        private int weight;
        public void setValue(int h, int w){
                height = h;
                weight = w;
        }
        public int getArea(){
                return height*weight;
        }
        public static void main(String[] args) {
                Rectangle r = new Rectangle();
```

```
                r.setValue(10, 15);
                System.out.println("Area: " + r.getArea());
        }
}
```

Output is-

```
Area: 150
```

Write a code from UML:

Rectangle
- height: int - width: int
+ setData (int, int): void + display (): void

```
public class Rectangle {
        private int height;
        private int weight;
        public void setValue(int h, int w) {
                height = h;
                weight = w;
        }
        public void display() {
                System.out.println(height + "," + weight);
        }
}
```

JAVA METHODS:

- A Java method is a collection of statements that are grouped together to perform an operation.

```
modifier returnType nameOfMethod (Parameter List)
{
  // method body
}
```

The syntax shown above includes –

- modifier – It defines the access type of the method and it is optional to use. Like can everyone see it or it have restricted access.
- returnType – Method may return a value.
- nameOfMethod – This is the method name. The method signature consists of the method name and the parameter list.
- Parameter List – The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- method body – The method body defines what the method does with the statements.

e.g.

```
public class Summation {

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = add(a, b);         //calling method
      System.out.println("Summation = " + c);
   }

   /** returns the summation of two numbers */

   public int add(int n1, int n2) //here modifier is "public",return type id "int", name
is "add",and "n1","n2" are parameter list.
{
      //this is method body
      sum=n1+n2;
      return sum;
   }
}
```

Method which will not return anything:

- we use void keyword as return type in this kind of method.

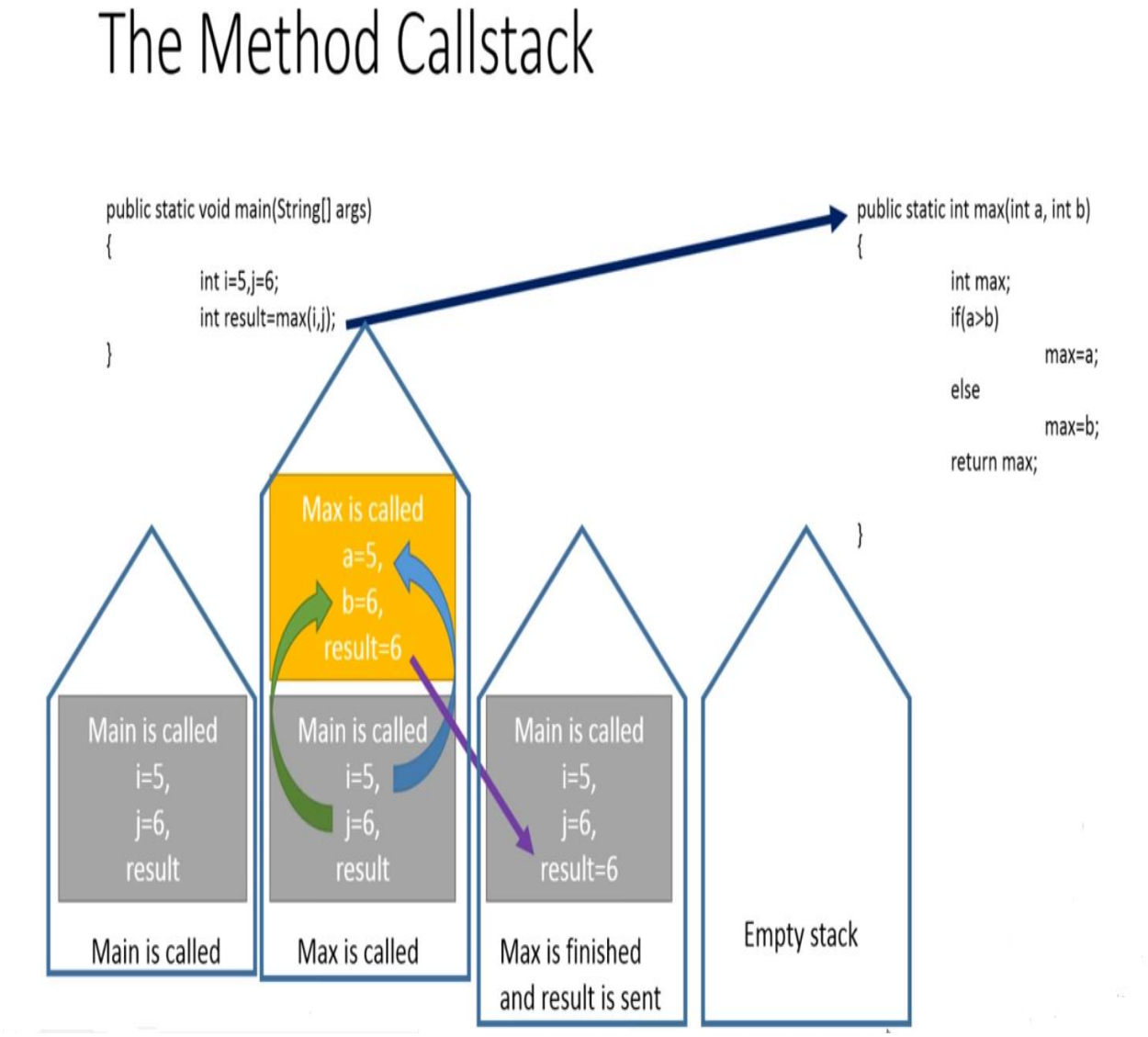
```
public void add(a,b)
```

```
{
   System.out.println("a+b ="+(a+b));
}
```

Method which will return value:

```
public int add(a,b)
{
   System.out.println("a+b ="+(a+b));

   return a+b;
}
```



METHOD OVERLOADING

Method within the same class that have the same name but slightly different characteristics(signature) like different number of parameters, different type of parameters that perform related operation are called overloaded methods.

```
public class Math
```

```
{
   public int add(int a,int b)
   {
      return a+b;
   }

   public double add(double a, double b) // changing data type
   {
      return a+b;
   }

   public int add(int a, int b, int c) // changing no. of arguments
   {
      return a+b+c;
   }

   public static void main(String[] args)
   {
      System.out.println(Math.add(11,11));

      System.out.println(Math.add(12.3,12.6));

      System.out.println(Math.add(12,13,15));
   }
}
```

- when you give two or more methods the same name within the same class, you are overloading the method name.
- Different method definitions have something different about their parameter list.
- Java distinguishes overloaded methods according to the number of parameters and the types of the parameters.
- When Java code is written to call a method, the compiler checks if the first argument has the same type as the first parameter, the second argument has the same type as the second parameter.
- If there is no such match, Java will try some simple type conversions[only smaller to bigger data type] such as casting into a double, to see whether that produces a match.
- It can be applied to void methods, to methods that return a value, to static methods, to non-static methods, and also constructors.
- The parameter list must be different for overloaded methods.

- You cannot use return type or access modifier alone to create overloaded methods.

Overloaded method causing ambiguous invocation lead to compiler errors.