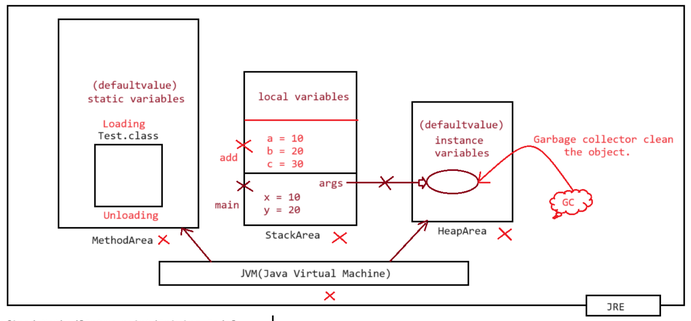


```
//Pre-Defined Method[Entry point/Driving Code]
public static void main(String[] args)
{
        int i =0;  <= main block
        for (int j = 0; j<3;j++ )  <= loop block
        {
            i = i+j;
        }
        System.out.println(i + " " + j);
}
                                   Not Accessible
```

```
//Pre-Defined Method[Entry point/Driving Code]
public static void main(String[] args)
{
        int x = 10;
        int y = 20;

        add(x,y);
}

static void add(int a, int b)
{
        int c = a+b;
        System.out.println(c);
}
```

```
class Calculator
{
    public void add(int a, int b)
    {
        System.out.println(a+b);
    }
    public void add(int a,int b, int c)
    {
        System.out.println(a+b+c);
    }
    public void add(double a, double b)
    {
        System.out.println(a+b);
    }
    public void add(float a, float b)
    {
        System.out.println(a+b);
    }
}
```

10,20

10,20,30

25.5,35.5

35.5f, 36.5f

Overloaded
1(add) : (M)Many Tasks
Outsider — 4 methods add

10,20
10,20,30
25.5,35.5
35.5f, 36.5f

Polymorphism
a. 1 thing in many forms

Technically
a. CompileTime Polymorphism
b. Runtime Polymorphism

Developer — Method-Overloading

1. Write methodName same for multiple tasks.
2. Change the Argument types.

```
class Test
{
    public static void main(String[] args)
    {
        Calculator c = new Calculator();
        c.add(10,20);
        c.add(10,20,30);
        c.add(20.5,30.5);
        c.add(20.5f,30.5f);
    }
}
```

```
class Calculator
{
    public void add(int a, int b)
    {
        System.out.println(a+b);
    }
    public void add(int a,int b, int c)
    {
        System.out.println(a+b+c);
    }
    public void add(double a, double b)
    {
        System.out.println(a+b);
    }
    public void add(float a, float b)
    {
        System.out.println(a+b);
    }
}
```

binding
binding
binding
binding

Compiler

1.No of Arguments
2.Datatype of Arguments

EarlyBinding

CompileTime
Polymorphism

Method Overloading → JVM → Output

Typecasting[Widening]

byte-----> short ------>int ------> long ------> float ----> double

char

```
class Test
{
    //overloaded method
    public void methodOne(int i){
        System.out.println("int arg method");
    }
    public void methodOne(float f){
        System.out.println("float arg method");
    }
}
```

```
public static void main(String[] args)
    {
        Test t= new Test();
        t.methodOne('a');  t.methodOne(10L);
    }
```

char — long

1. search for char argument type only[Exact match]
2. for the arugment type, perform typepromotion(typecasting) to its best for binding.
3. After step2, if compiler is not able to perform binding, then it would result in "Compile-Time Error".