Programming Paradigms:

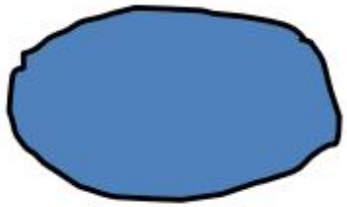
1. Unstructured Style
2. Structured/Procedural/Functional Style
3. Object-Oriented Style

Functions in C

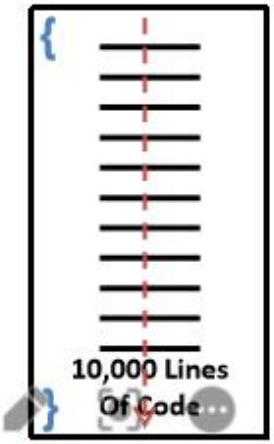
Need of Functions:

UNSTRUCTURED STYLE

Small Task

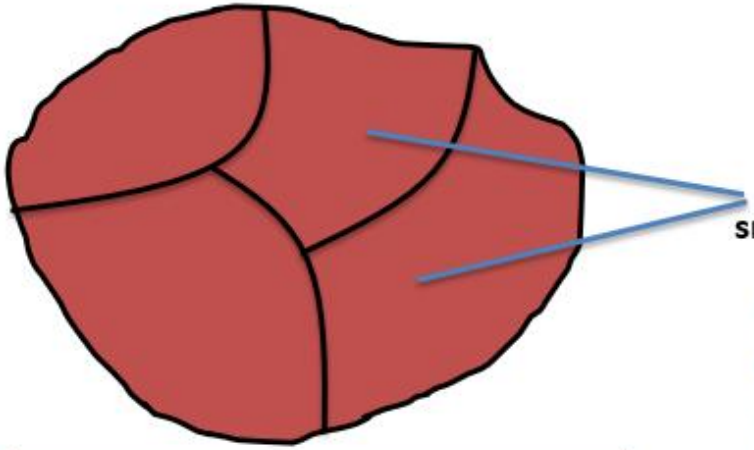


Unstructured Programming/
Non-Structured Programming



STRUCTURED STYLE

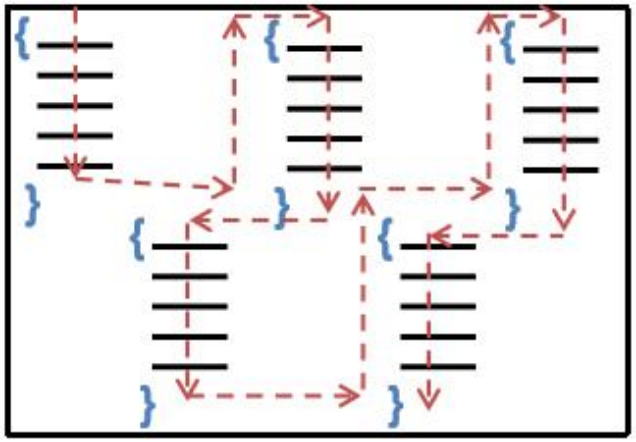
Large Task



Divide it into
smaller sub-tasks

- Function
- Procedure
- Sub-program
- Module
- Sub-module
- Routine
- Sub-routine
- Method

Structured Programming/
Modular Programming/
Procedure-Oriented Programming/
Procedural Programming/
Functional Programming/...



Syntax of Functions:

A Function must have the following 4 components:

1. **Name** of the function
2. **Input** to the function (*optional)
3. **Activity** performed by the function
4. **Output** from the function

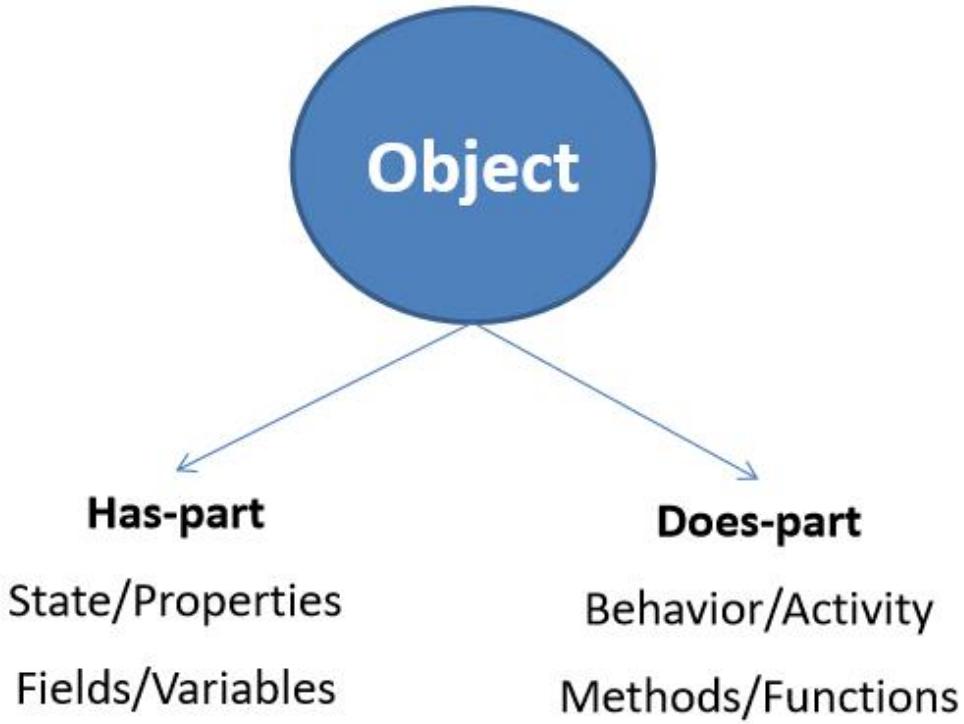
```
output name(input)
{
    //activity or task
}
```

```
return_type name(parameters)
{
    //body of the function
}
```

2 Major Advantages of Functions:

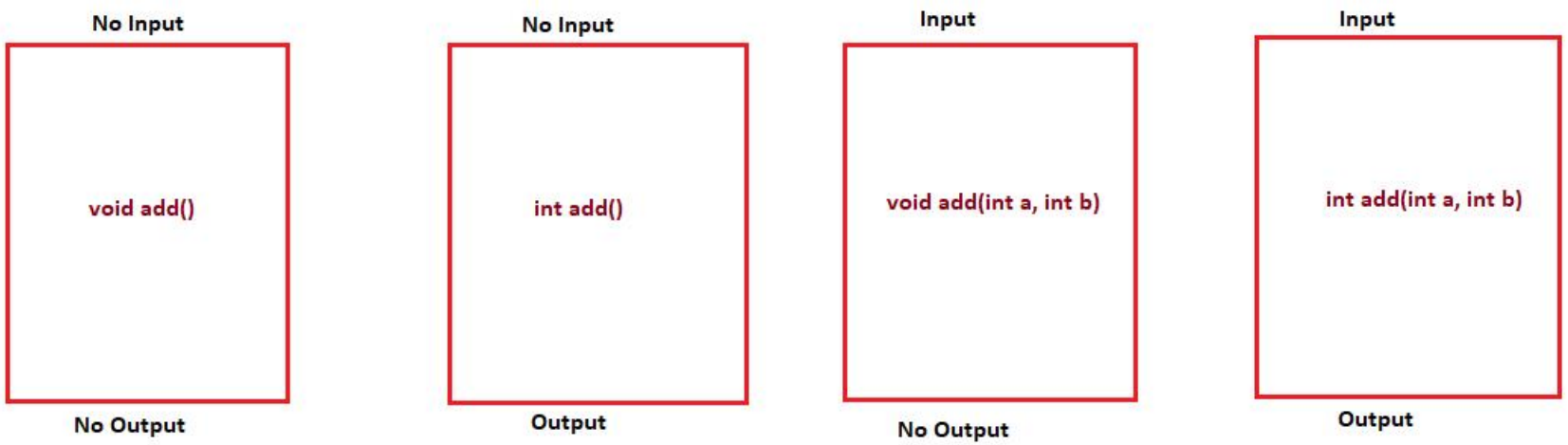
1. **Modularity**
2. **Reusability**

Methods in Java

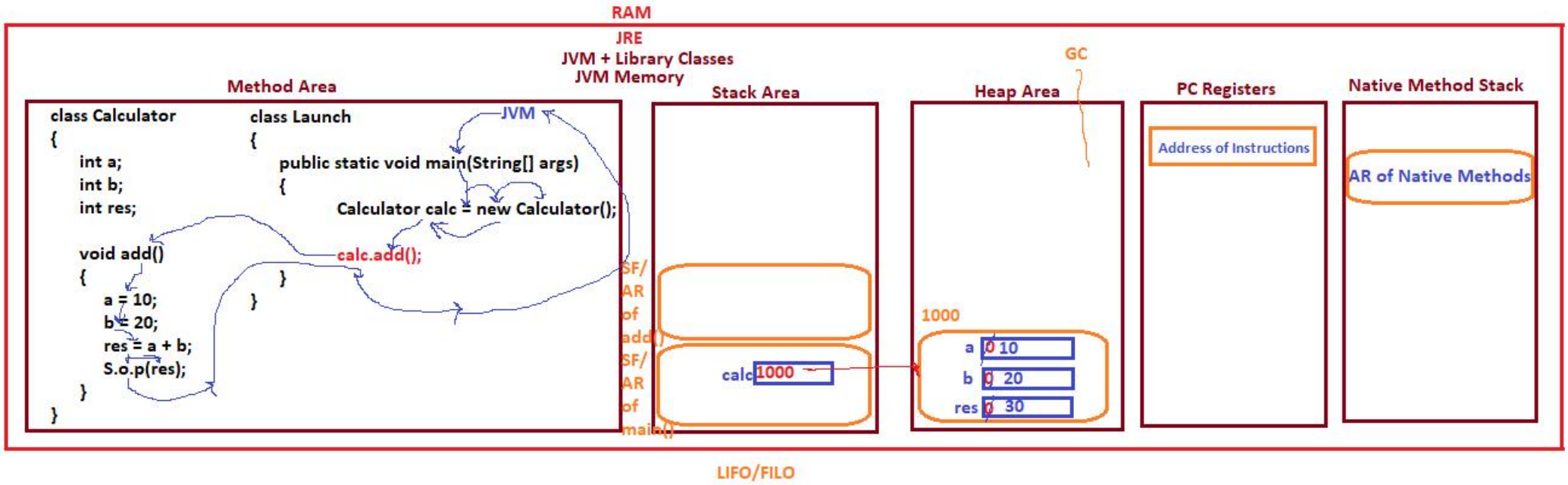


1. A method, like a function, is a block of code or collection of statements grouped together to perform a certain task or operation. The difference is that a method is associated with an object, while a function is not.
2. In Java, methods are used to implement the behavior of the object.
3. A method must always be declared within a class.
4. A method can directly access all the instance variables of the enclosing class.

Types of Methods:



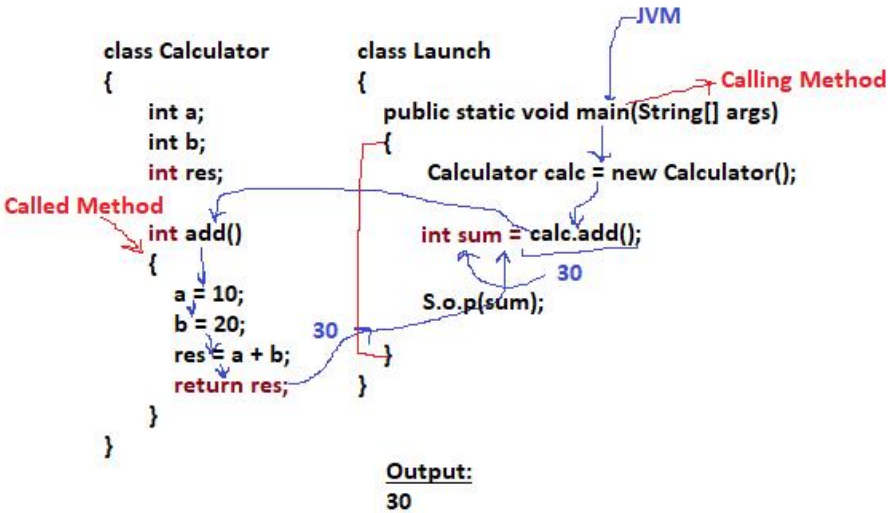
Type-1: Methods with no parameters and no return value



Output:

30

Type-2: Methods with no parameters but with a return value



Output:

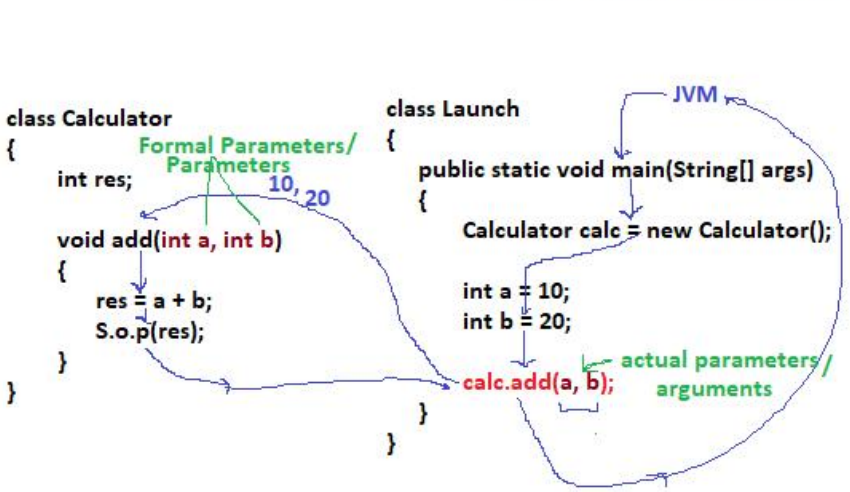
30

type variable;

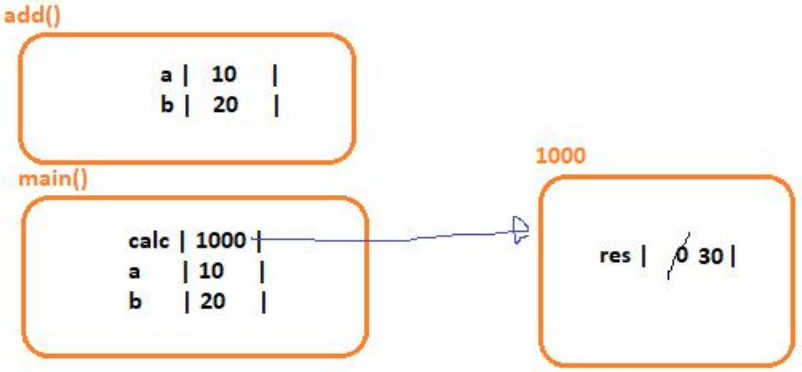
Calculator calc; <- Reference Varianle

int sum; <- Primitive Variable

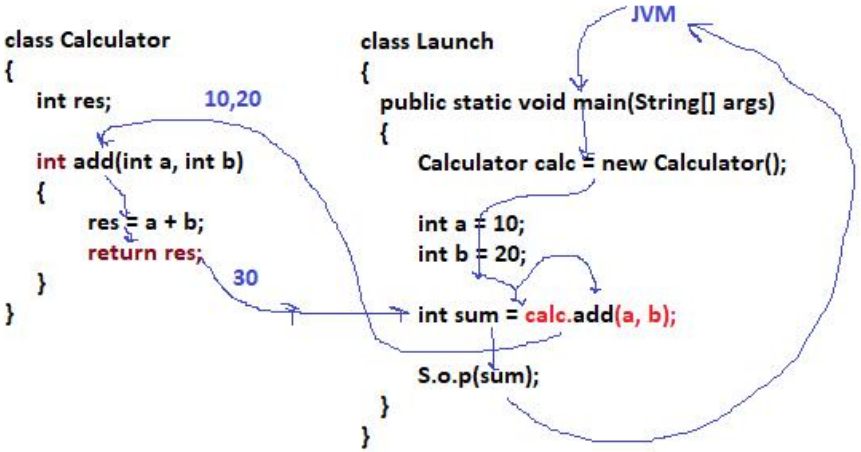
Type-3: Methods with parameters but no return value



Output:
30



Type-4: Methods with parameters and return value



Output:

30

C Language:

Method Overloading promotes Virtual Polymorphism

```
#include<stdio.h>
int add1(int a, int b)
{
    return  a + b;
}

float add2(int a, float b)
{
    return a + b;
}

float add3(float a, float b)
{
    return a + b;
}

float add4(float a, int b)
{
    return a + b;
}

double add5(double a, double b)
{
    return a + b;
}

double add6(int a, double b)
{
    return a + b;
}

double add7(float a, double b)
{
    return a + b;
}

int add8(int a, int b, int c)
{
    return a + b + c;
}

float add9(float a, float b, float c)
{
    return a + b + c;
}

double add10(double a, double b, double c)
{
    return a + b + c;
}

double add11(int a, float b, double c)
{
    return a + b + c;
}

double add12(int a, float b, float c, double d)
{
    return a + b + c + d;
}

void main()
{
    int a = 10, b = 20, c = 30;
    float p = 10.5, q = 20.5, r = 30.5;
    double x = 100.55, y = 200.55, z = 300.55;

    printf("%lf", add12(a, p, q, x));
    printf("%d", add8(a, b, c));
    printf("%lf", add5(x, y));
    printf("%f", add2(a, p));

}
```

C --> C++ ---------> JAVA {C++ ++ --}

[Function Overloading] [Method Overloading]

Method Overloading promotes Virtual Polymorphism

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

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

  float add  (float a, float b)
  {
      return a + b;
  }

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

  double add  (double a, double b)
  {
        return a + b;
  }

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

  double add  (float a, double b)
  {
      return a + b;
  }

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

  float add  (float a, float b, float c)
  {
       return a + b + c;
  }

  double add  (double a, double b, double c)
  {
      return a + b + c;
  }

  double add  (int a, float b, double c)
  {
      return a + b + c;
  }

  double add  (int a, float b, float c, double d)
  {
      return a + b + c + d;
  }
}
```

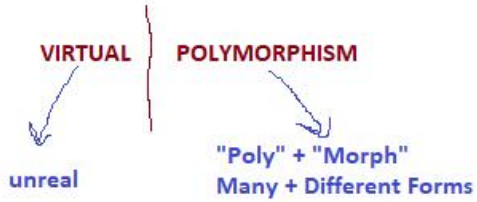
```
class Launch
{
    p s v main(...)
    {
        int a = 10, b = 20,  c = 30;
        float p = 10.5f, q = 20.5f, r = 30.5f;
        double x = 100.55, y = 200.55, z = 300.55;

        Calculator calc = new Calculator();

        S.o.p(calc.add(a, p, q, x));
        S.o.p(calc.add(a, b, c));
        S.o.p(calc.add(x, y));
        S.o.p(calc.add(a, p));

    }
}
```

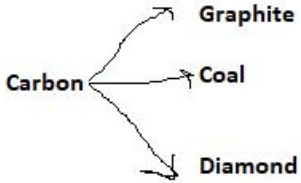
0. Name of the Method
1. Number of Parameters
2. Data Types of Parameters
3. Sequence of Data Types of Parameters



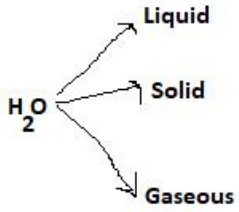
1:M

~~M:M~~

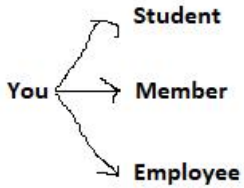
1:1



1:Many



1:Many



1:Many