Methods In Java

* What is method?
  - Method is used to perform a certain task
  - Collection of instructions that performs a specific task
  - It can be used to bring the code readability & re-usability

  For eg:-

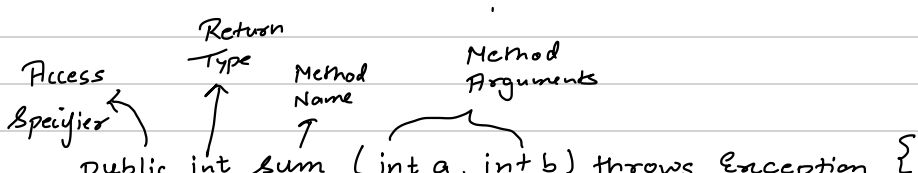
```java
public class Calculation {

    public int sum(int val1, int val2){
        int  total = val1 + val2;          → Instruction 1
        //doing some logging stuff         → Instruction 2
        System.out.println("addition of val1 and val2 is:" + total);
        return total;                      → Instruction 3
    }

    public int getPriceOfPen(){
        int capPrice = 2;
        int penBodyPrice = 5;
        int totalPenPrice = sum(capPrice, penBodyPrice);
        return totalPenPrice;
    }
```

Here sum is a method which will take 2 arguments & return the addition of both the numbers. Since we reused the code of method to perform add" so it provides reusability.

* How to declare method?



* Access Specifiers
  - Defines accessibility of a method i.e who can use the method
  - There are 4 types of access specifiers in Java methods:-

1) Public : can be accessed through any class in any package.
2) Private : can be accessed by methods only in the same class
3) Protected : can be accessed by other classes in same package or other sub classes in different package.
4) Default : It can only be accessed by classes in same package. If we do not mention anything, then Default access specifier is used by Java.

* Return Type
  - It tells what type the method will return after computation. If the method don't return anything, void return type is used.
  - Use class name or primitive data types as return type of the method.

* Method Name :
  - It should be verb (or some kind of action)
  - Should start with small letter and follow camel case in case of multiple words

* Method Parameters
  - It's a list of variables that will be used in the method.
  - Parameter list can be blank too.

* Method Body :
  - Method body get finished when you call 'return' in mid.
  - Gets finished when reached to the end
  - We can also stop method by 'return' even for void return type

⇒ Types Of Methods :-

* System Defined Methods :-
  - Methods which are already defined and ready to use in Java like Math.sqrt()

* User Defined Methods :
  - Methods which the programmer create based upon the program necessity

* Overloaded Method :
  - More than one method with same name is created in same class
  - Overloaded Method only gets differentiated based on arguments so name should be same, arguments should be differend and return type is not even considered.

```java
public class Invoice {

    void getInvoice() {
        System.out.println("inside invoice method");
        return;
    }

    void getInvoice(String z){

    }

    int getInvoice(int a){

        //doing some other stuff
    }

    void getInvoice(int a, int b){

        //something else
    }

    public void printInvoice(){
        getInvoice();
    }
}
```

* Overridden Method
- Subclass / Child Class has the same method as the parent class.

* Static Methods:
- These methods are associated with the class
- Can be called just with class name.
- Static methods can not access Non Static Instance variables and methods
- Static methods cannot be overridden.

So when to declare method static:
- Methods which do not modify the state of the object can be declared static.
- Utility method which do not use any instance variable and compute only on arguments.
  Example: Factory design pattern

* Final Methods:
- Final Method cannot be overridden in Java. It is so because final method means its implementation cannot be changed. If child class cannot change its implementation then no use of overridden.

* Abstract Method
- It is defined only in abstract class.
- Only method declaration is done.
- Its implementation is done in child classes

* Variable Arguments (Varargs):
  - Variable number of inputs in the parameter.
  - Only one variable argument can be present in the method.
  - It should be the last argument in the list.
  - Used when we don't know the number of arguments

For Eg:-

```
public class Calculation {

    static int carPrice = 40;

    public int sum(int a, int ...variable){

        int output = 0;
        for(int var : variable){
            output = output + var;
        }
        return output;
    }

}
```

Variable argument

So while calling, we can give multiple no. of arguments.

```
public class Main {

    public static void main(String args[]) {

    Calculation calculationObj = new Calculation();
    calculationObj.sum( a: 3);
    calculationObj.sum( a: 3, ...variable: 8,9,10);
    calculationObj.sum( a: 3, ...variable: 8,9,10,3,3,2,2,2,2,2,2);


    }
}
```

CONSTRUCTORS IN JAVA

* What is constructor ?
  - It is used to create an instance / initialise the instance variable
  - It's similar to method except :
    - Name : Constructor name is same as class name
    - Return Type : Constructor do not have any return type.
    - Constructor cannot be static or final or abstract, synchronized.

* new keyword tells java to call constructor

* Why constructor name is same as of class name ?
  - Constructor name is always same as class name because it is easy to identify & there is no return type because implicitly java adds class as return type.

* Why constructor do not have return type ?
  - There can be methods with same name and even class as return type but they cannot be called constructors as they do not obey the rules of constructor ie same name without return type.

* Why constructor cannot be final ?
  - Constructors are different from usual methods & cannot be inherited. So it doesn't make sense to make them final because final is used to prevent overriding & If constructors cannot be inherited then

there is no requirement for final

* Why constructor cannot be abstract?
- Since for abstract method, the responsibility of implementation is of child class. But constructors can't even be inherited so no point of making them abstract.

* Why constructor cannot be static?
- Since static methods can only access static variables & other static methods, so it won't be able to initialise the instance variable. We also won't be able to use constructor chaining & call super().

* Can we define constructor in interface?
- No because we cannot create object so no point of constructor.

⇒ Type Of Constructors :-

1) Default Constructor
- When we do not define a constructor, java internally provides a constructor which is known as default constructor. Default constructor also set default values for all the instance variables. It is added only when we do not define a constructor

2) No Argument Constructor
- A constructor that does not take any argument. It is very similar to default constructor but we are defining it instead of java.

3) Parameterised Constructors

- It takes arguments and assign the instance variables with those parameters. We can initialise one or multiple instance variables using a parameterised constructor. For the variables where we don't provide any argument, they'll be instantiated with default values.

4) Constructor overload

- We can create multiple constructor with different parameteres.

5) Private Constructor

- We can create a private constructor & no one outside the class will be able to call the constructor. This is used usually in Singleton design pattern. To create an object of a class having private constructor, we can create another static method to create the object & then call that method using class name.

⇒ Constructor Chaining

* It means that we can call one constructor in other constructor. This is done using this() & super(). To chain a constructor within the same class, this() is used. For eg:

Calculation.java | Calculation.class | Main.java

```
public class Calculation {

    String name;
    int empID;

    Calculation(){
        this( empID: 10);
    }


    Calculation(int empID){

        this( name: "sj", empID);
    }

    Calculation(String name, int empID){

        this.name = name;
        this.empID = empID;
    }

```

Here we called other constructor within a constructor using this().

* Using super()
  - The constructor of a child class always invokes the constructor of parent class first & then invokes its own constructer. This is done using super(), so if we explicitly don't add super() in child constructor then Java adds it internally

So this() & super() are used for constructor chaining.

* If the parent class has a parameterised constructor, then we'll have to mandatorily pass an argument to super() to call the parent class's parameterised constructor.