

Static Method: A method applies to the class in which it's declared as a whole and is known as a static method or a class method. Ex:

```
2
3 ► public class Main {
4
5 ►   public static void main(String[] args) {
6     call();
7   }
8   public static void call(){ // static method
9     System.out.println("Hello");
10  }
11 }
12 |
```

Static Variable: It is also known as class variable which belongs to the class and which is not unique for each object and gets only one memory address.

```
2
3 ► public class Main {
4
5 ►   public static void main(String[] args) {
6     System.out.println(ClsVar.id);
7   }
8 }
9
10 class ClsVar{
11     static int id = 211211042;
12 }
```

Instance Variable: It is also known as non-static variable which is unique for each object and get different memory for different object.

```

2
3 ▶ public class Main {
4
5 ▶     public static void main(String[] args) {
6         InsVar x = new InsVar();
7         System.out.println(x.id);
8     }
9 }
10
11 class InsVar{
12     int id = 211211042;
13 }
14

```

Points:

1. Static method can be called without creating an instance of the class.
2. We can't call non static method inside static method.
3. We can use class variable inside static method and non-static method.
4. We can use instance variable inside non static method but can't use static method.
5. Static block executed before the main method.

```
2
3 ▶ public class Main {
4   static{
5       System.out.println("I am a");
6   }
7 ▶ public static void main(String[] args) {
8       System.out.println("Student");
9   }
10 }
11
12
13 |
```

Static can be:

1. Variable
2. Method
3. Block
4. Nested Class

public static void main(String[] args) Meaning?

Public: It is access modifier of the main method. It has to be public so java compile can call this method. Non-public not allowed.

Static: static method means it can be called without creating an object.

Void: It is main method return type. Void means nothing. Main method doesn't return anything.

Main: it is name of the method. It is fixed and java program execute from main method. It is starting point.

String [] args: It is main method parameter of string array. This is called command line argument.

Constructor: It is a special method that is called when an object of a class is created.

There are two types of constructor:

1. Default constructor
2. Parameterized constructor

Default constructor: This type of constructor doesn't have any parameter and if we don't create any constructor the java compiler will automatically create default constructor. Default constructor can be initialized with any uninitialized instance variable with default values.

```
2
3 ▶ public class Main {
4 ▶     public static void main(String[] args){
5         DefCons obj = new DefCons();
6         System.out.println(obj.id);
7     }
8 }
9
10 class DefCons{
11     int id;
12
13     DefCons(){ // default constructor
14         id = 0;
15     }
16 }
```

Parameterized constructor: It can take one or more parameters. Such constructors are known as parameterized constructors.

```

2
3 ► public class Main {
4 ►     public static void main(String[] args){
5         DefCons obj = new DefCons( sid: 10);
6         System.out.println(obj.id);
7     }
8 }
9
10 class DefCons{
11     int id;
12     DefCons(int sid){
13         id = sid;
14     }
15 }

```

Method Overloading: It is multiple methods with same name but different parameter.

```

2
3 ► public class Main {
4 ►     public static void main(String[] args){
5         add( x: 2, y: 5);
6         add( x: 2.3, y: 5.5);
7     }
8
9     public static void add(int x,int y){
10         System.out.println(x + y);
11     }
12
13     public static void add(double x,double y){
14         System.out.println(x + y);
15     }
16 }
17

```

Constructor Overloading: Having more than one constructor with different parameter is called constructor overloading.

```
2
3 ▶ public class Main {
4 ▶     public static void main(String[] args){
5         Student ob1 = new Student();
6         Student ob2 = new Student(n: "Fahim");
7
8         System.out.println(ob1.name + " " + ob2.name);
9     }
10
11 }
12
13 class Student{
14     String name;
15
16     Student(){
17         name = null;
18     }
19     Student(String n){
20         name = n;
21     }
22 }
```

Encapsulation: Data hiding and data binding is called encapsulation.

```

2
3  ▶ public class Main {
4  ▶      public static void main(String[] args){
5          Student s1 = new Student();
6          s1.setName("Fahim");
7      }
8
9  }
10
11  class Student{
12      private String name;
13
14      public void setName(String n) {
15          name = n;
16      }
17  }
18
19  |

```

Accessors: It is a method known as get method or getter.

Mutator: It is a method known as set method or setter.

```

2
3  ▶ public class Main {
4  ▶      public static void main(String[] args){
5          Student s1 = new Student();
6          s1.setName("Fahim");
7          System.out.println(s1.getName());
8      }
9
10 }
11
12 class Student{
13     private String name;
14
15     public void setName(String n) {
16         name = n;
17     }
18
19     public String getName(){
20         return name;
21     }
22 }
23

```

Inheritance: It is a relationship between two or more classes where derived class inherits properties of pre-existing classes.

Inheritance is the procedure in which one class inherits the attributes and methods of another class.

Super Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Types of inheritance:

1. **Single inheritance**
2. **Multilevel inheritance**
3. **Hierarchical inheritance**
4. **Multiple Inheritance (It doesn't support java)**
5. **Hybrid Inheritance**


```
3
4 // Single Inheritance
5 public class Basic {
6     public static void main(String[] args){
7         Dog d = new Dog();
8         d.food = "Bread";
9         d.eat();
10    }
11
12 }
13
14 class Animal{
15     public String food;
16     public void eat(){
17         System.out.println("Eating " + food);
18     }
19 }
20
21 class Dog extends Animal{
22     public void berk(){
23         System.out.println("Barking");
24     }
25 }
26
```

```
1
2 package com.mycompany.java_gui;
3
4 // Multilevel Inheritance
5 public class Basic {
6     public static void main(String[] args){
7         BabyDog b = new BabyDog();
8         b.eat();
9     }
10 }
11
12
13
14 class Animal{
15     public void eat(){
16         System.out.println("Eating");
17     }
18 }
19
20 class Dog extends Animal{
21     public void berk(){
22         System.out.println("Barking");
23     }
24 }
25
26 class BabyDog extends Dog{
27     public void weep(){
28         System.out.println("Weeping");
29     }
30 }
```

```

3
4 // Hierarchical Inheritance
5 public class Basic {
6     public static void main(String[] args){
7         BabyDog b = new BabyDog();
8         b.eat();
9     }
10 }
11
12 class Animal{
13     public void eat(){
14         System.out.println("Eating");
15     }
16 }
17
18 class Dog extends Animal{
19     public void berk(){
20         System.out.println("Barking");
21     }
22 }
23
24 class BabyDog extends Animal{
25     public void weep(){
26         System.out.println("Weeping");
27     }
28 }
29

```

Method Overriding: If subclass has the same method as declared in the parent class, it is known as method overriding in java.

Rules:

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. Static method cannot be overridden

```

3
4 // Method Overriding
5
6 public class Basic {
7     public static void main(String[] args){
8         Bike b = new Bike();
9         b.run(5);
10        Vehicle v = new Vehicle();
11        v.run(10);
12    }
13 }
14
15 class Vehicle{
16
17     public void run(int a){
18         System.out.println("Vehicle is running at " + a);
19     }
20
21 }
22
23 class Bike extends Vehicle{
24
25     @Override
26     public void run(int a){
27         System.out.println("Bike is running at " + a);
28     }
29 }
30
31

```

This (keyword): It is a reference variable that refers to the current object.

```

5
6 public class Basic {
7     public static void main(String[] args){
8         Vehicle v = new Vehicle();
9         v.show();
10    }
11 }
12
13 class Vehicle{
14     private int price;
15     private String name;
16
17     Vehicle(){
18         this(0,"Null");
19     }
20
21     Vehicle(int price,String name){
22         this.price = price;
23         this.name = name;
24     }
25
26     public void show(){
27         System.out.println(price + " " + name);
28     }
29 }

```

Super (Keyword): The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Rules:

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super () can be used to invoke immediate parent class constructor.
4. We can't inherit super class constructor in child class.

```

4 // Variable
5
6 public class Basic {
7     public static void main(String[] args){
8         Car c = new Car();
9         c.show();
10    }
11 }
12
13 class Vehicle{
14     String color = "Black";
15 }
16
17 class Car extends Vehicle{
18     String color = "Red";
19
20     public void show(){
21         System.out.println(super.color);
22     }
23 }

```

```

3
4 // Method
5
6 public class Basic {
7     public static void main(String[] args){
8         Car c = new Car();
9         c.color();
10    }
11 }
12
13 class Vehicle{
14     public void color(){
15         System.out.println("I am color");
16     }
17 }
18
19 class Car extends Vehicle{
20
21     @Override
22     public void color(){
23         super.color();
24     }
25 }
26
27

```

```
4 // Method
5
6 public class Basic {
7     public static void main(String[] args){
8         Car c = new Car();
9     }
10 }
11
12 class Vehicle{
13
14     Vehicle(){
15         System.out.println("I am Vehicle class"); // 1st show
16     }
17 }
18
19 class Car extends Vehicle{
20
21     Car(){
22         System.out.println("I am car class"); // 2nd show
23     }
24 }
25
```

```

3
4 // Constructor
5
6 public class Basic {
7     public static void main(String[] args){
8         Car c = new Car("Boss", "Toyota", 2000);
9         c.show();
10    }
11 }
12
13 class Vehicle{
14     protected String name;
15     protected String brand;
16
17     Vehicle(String name, String brand){
18         this.name = name;
19         this.brand = brand;
20     }
21 }
22
23 class Car extends Vehicle{
24
25     private int year;
26
27     Car(String name, String brand, int year){
28         super(name, brand);
29         this.year = year;
30     }
31
32     public void show(){
33         System.out.println(name + " " + brand + " " + year);
34     }
35 }
36
37

```

Abstraction: It is a process of hiding the implementation details and showing only functionality to the user.

There are two way to achieve abstraction in java

1. Abstract class
2. Interface

Point:

1. Abstract class must be declared with abstract keyword
2. Abstract class can have abstract and non-abstract method
3. It can't be used to create objects to access it must be inherited from another class
4. Abstract method doesn't have body in abstract class. The body provided by the subclass
5. An abstract class can have both abstract and regular or concrete method
6. It can have constructor and static method also.

```
3
4 public class Main {
5     public static void main(String[] args){
6         Dog d = new Dog();
7
8         d.animalsound();
9         d.sleep();
10    }
11 }
12
13 abstract class Animal{
14     public abstract void animalsound(); // Abstract method
15
16     public void sleep(){ // Regular or concrete method
17         System.out.println("ZZZZzzz");
18     }
19 }
20
21 class Dog extends Animal{
22
23     @Override
24     public void animalsound(){ // abstract method body
25         System.out.println("Ghaw Ghaw");
26     }
27 }
28
```

```

3
4 public class Main {
5     public static void main(String[] args){
6         Add a = new Add(5,10);
7         System.out.println(a.add());
8     }
9 }
10
11
12 ① abstract class Sum{
13     int num1; // Data Field
14     int num2;
15
16     Sum(int num1,int num2){ // Constructor
17         this.num1 = num1;
18         this.num2 = num2;
19     }
20
21     ② abstract public int add();
22 }
23
24
25 class Add extends Sum{
26
27     Add(int num1,int num2){
28         super(num1,num2);
29     }
30
31     @Override
32     ③ public int add() {
33         return num1 + num2;
34     }
35
36 }
37

```

Interface: It is completely abstract class that is used to group related method with empty body.

Points:

1. It is used to achieve abstraction
2. It support multiple inheritance
3. It doesn't have a constructor.
4. Interface method default abstract and public

5. Interface attributes are by default public, static and final

```
3
4 public class Main {
5     public static void main(String[] args){
6         Dog d = new Dog();
7         d.sound();
8     }
9 }
10
11 interface Animal{
12     public void sound();
13 }
14
15 class Dog implements Animal{
16
17     @Override
18     public void sound(){
19
20     }
21 }
22
23
```

```
3
4 public class Main {
5     public static void main(String[] args){
6         Dog d = new Dog();
7         d.name();
8         d.sound();
9     }
10 }
11
12 interface Animal1{
13     public void name();
14 }
15
16 interface Animal2{
17     public void sound();
18 }
19
20 class Dog implements Animal1,Animal2{
21
22     @Override
23     public void sound(){
24         System.out.println("lolu");
25     }
26
27     @Override
28     public void name() {
29         System.out.println("Dog");
30     }
31 }
32
33
```

```
3
4
5
6 public class Main {
7     public static void main(String[] args){
8         C prr = new C();
9         prr.m();
10        prr.n();
11    }
12 }
13
14 interface a{
15     public void m();
16 }
17
18 interface b extends a{
19     public void n();
20 }
21
22 class C implements b{
23
24     @Override
25     public void m(){
26         System.out.println("M");
27     }
28
29     @Override
30     public void n(){
31         System.out.println("N");
32     }
33 }
34
35
```

Polymorphism: It means “many forms”. It is a concept which can perform a single action different ways.

There are two types of polymorphism

1. Compile-time polymorphism (This type of polymorphism is static and it is achieved by method overloading or operator overloading but java don't support the operator overloading).
2. Runtime polymorphism (Also known as dynamic method dispatch. It is achieved by method overriding)

Points:

1. Polymorphism in java performs by method overloading and method overriding.
2. Runtime polymorphism can't be achieved by data members.

```
2
3 // Compile time Polymorphism
4
5 public class Main {
6
7     public static int add(int a,int b){
8         return a + b;
9     }
10
11     public static double add(double a,double b){
12         return a + b;
13     }
14
15     public static void main(String[] args){
16         System.out.println(add(2,4));
17         System.out.println(add(2.3,2.2));
18     }
19
20 }
21
```

```

3 // Runtime time Polymorphism
4
5 public class Main {
6
7     public static void main(String[] args){
8         Bike b = new A(); // upcasting
9         b.run();
10        Bike c = new B();
11        c.run();
12    }
13
14 }
15
16 class Bike{
17     public void run(){
18         System.out.println("Running");
19     }
20 }
21
22 class A extends Bike{
23     @Override
24     public void run(){
25         System.out.println("Run 1");
26     }
27 }
28
29 class B extends Bike{
30     @Override
31     public void run(){
32         System.out.println("Run 2");
33     }
34 }
35

```

Up casting: Child class to parents class.



Down casting: Parents class to child class.

File Handling

```
3 // Create File
4
5 import java.io.File;
6 import java.io.IOException;
7
8 public class Main {
9
10     public static void main(String[] args){
11
12         try{
13             File myFile = new File("javafile.txt");
14
15             if(myFile.createNewFile()){
16                 System.out.println("File Created : " + myFile.getName());
17             }
18             else{
19                 System.out.println("File already exists");
20             }
21         }
22         catch(IOException e){
23             System.out.println("An Error Occurred");
24             e.printStackTrace();
25         }
26     }
27 }
28
29 }
30
```



```
2
3 // Read from file
4
5 import java.io.*;
6 import java.util.Scanner;
7
8 public class Main {
9
10 public static void main(String[] args) {
11     try{
12         File f = new File("javaFile.txt");
13         Scanner sc = new Scanner(f);
14
15         while(sc.hasNext()){
16             String contant = sc.nextLine();
17             System.out.println(contant);
18         }
19     }
20     catch(IOException e){
21         e.printStackTrace();
22     }
23 }
24 }
25
```

```
2
3 // Write to File
4
5 import java.io.FileWriter;
6
7 import java.io.IOException;
8
9 public class Main {
10
11     public static void main(String[] args){
12
13         try{
14              FileWriter myfile = new FileWriter("javafile.txt");
15             myfile.write("Hi! Boss");
16             myfile.close();
17             System.out.println("Write successfully");
18
19         }
20         catch(IOException e){
21             System.out.println("An Error Occurred");
22              e.printStackTrace();
23         }
24     }
25 }
26
27
28
```

Exception Handling

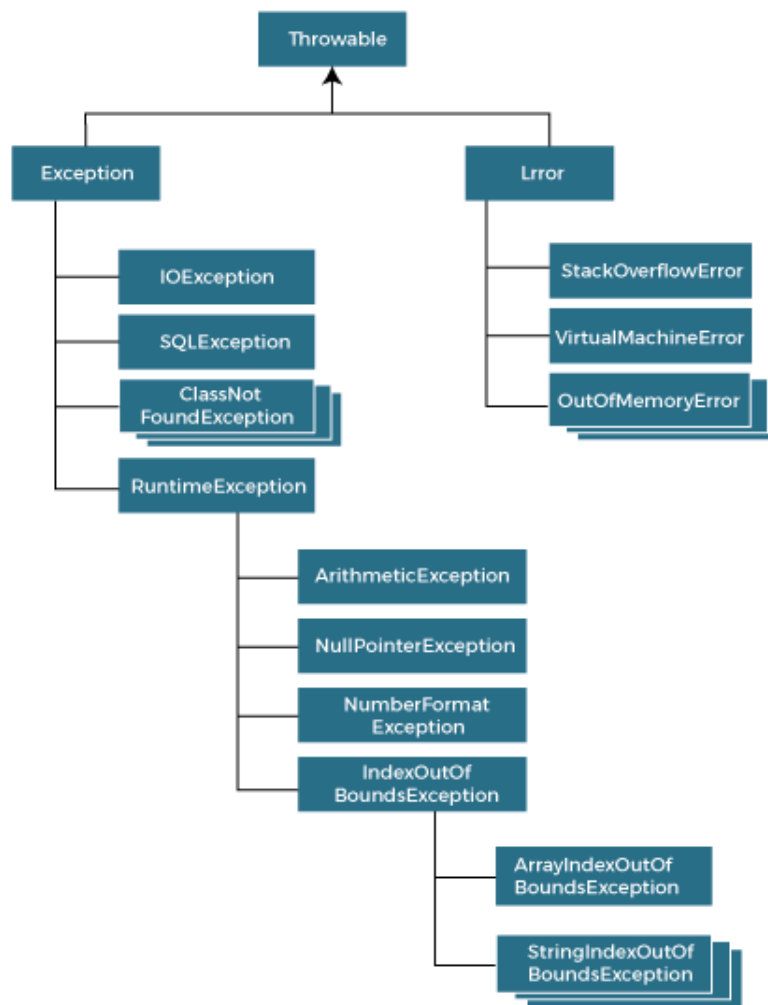
Exception handling: It is one of the powerful mechanisms to handle the runtime errors so that normal flow of the application can be maintained.

Types of Exception:

- 1. Checked (Checked at compile time)**
- 2. Unchecked (Checked at run-time)**
- 3. Error**

Point:

1. Its handle runtime error
2. Throws keyword is used with method signature.
3. Throw keyword is used for custom exception.



```

7
8 public class Main {
9     public static void main(String[] args){
10         try{
11             int a = 100/0;
12         }
13         catch(ArithmeticException e){
14             System.out.println(e);
15             System.out.println("Rest Code...");
16         }
17         finally{
18             System.out.println("Hello");
19         }
20     }
21
22
23 }
24

```

```

7
8 public class Main {
9     public static void main(String[] args){
10         try{
11             int [] a = {1,2};
12             System.out.println(a[10]);
13         }
14         catch(ArithmeticException e){
15             System.out.println(e);
16         }
17         catch(Exception e){
18             System.out.println(e);
19             System.out.println("Exception");
20         }
21     }
22
23
24 }
25

```

```

7
8 public class Main {
9     public static void main(String[] args){
10
11         int a = 2;
12         int b = 3;
13
14         System.out.println("Before Exception");
15
16         if(a != b){
17             throw new ArrayIndexOutOfBoundsException("Index out of range");
18         }
19         else{
20             System.out.println("No Error");
21         }
22     }
23 }
24

```

```

8 import java.io.*;
9
10 public class Main {
11     public static void main(String[] args){
12         // checked exeception
13
14         FileWriter fw = new FileWriter("Java.txt");
15
16     }
17 }
18

```

This is checked exception. This type of exception forced by the compiler used to indicate exceptional condition that is out of the control of the program.

```

7
8 public class Main {
9     public static void main(String[] args){
10         // Unchecked exeception
11
12         int a = 10 / 0;
13
14     }
15 }
16

```

Thread

Thread allows a program to operate more efficiently by doing multiple things at the same time. Unit of process.

```
7
8
9 public class Main {
10     public static void main(String[] args){
11         System.out.println(Thread.activeCount()); // how many active thread
12         System.out.println(Thread.currentThread().getName()); // thread name
13     }
14 }
15
```

```
7
8 public class Main {
9     public static void main(String[] args){
10
11         for(int i = 0 ;i < 5;i++){
12             try {
13                 System.out.println(i);
14                 Thread.sleep(1000);
15             } catch (InterruptedException ex) {
16                 System.out.println(ex);
17             }
18         }
19     }
20 }
21
22
```

```

7
8 public class Main {
9     public static void main(String[] args){
10
11         thread a = new thread();
12         a.start();
13
14         for(int i = 0;i < 10;i++){
15             System.out.println("Iam not Thread : "+ i);
16         }
17
18     }
19 }
20

```

com.mycompany.testproject.Main > main > for (inti = 0; i < 10; i++) >

Notifications Output - Run (TestProject) X

```

--- maven-compiler-plugin:3.1:compile (default-compile) @ TestProject ---
Nothing to compile - all classes are up to date

--- exec-maven-plugin:1.5.0:exec (default-cli) @ TestProject ---
Iam not Thread : 0
I am thread : 0
Iam not Thread : 1
I am thread : 1
Iam not Thread : 2
I am thread : 2
Iam not Thread : 3
I am thread : 3
Iam not Thread : 4
I am thread : 4
Iam not Thread : 5
I am thread : 5
Iam not Thread : 6
I am thread : 6
Iam not Thread : 7
Iam not Thread : 8
Iam not Thread : 9
I am thread : 7
I am thread : 8
I am thread : 9
-----

```