3-Static Keyword

In General:

▼ In General:

- Static is a non-access modifier in java
- Indicates that a member belongs to "class", rather than to "object".
- in Java, static is applicable for the followings:
  - a) Variables
  - b) Methods
  - c) Blocks

A) Variables:

▼ A) Variables:

- Variable: a name of specific location in memory.
- It's a container which holds the value.
  - Three kind of variables exist in java:
    - Local variable
    - Instance variable
    - Static variable

▼ 1-Local Variable:

- A variable declared inside a '**method**' or '**constructor**' or '**block**'.
- Accessible and visible only within the declared method, constructor, or block.
- Local variables can not have '**access modifier**'
- Local variables can not be '**static**'

- Initialization is mandatory for local variables

```
public class MyClass {


   //local variables are only visible within the method
   public static void main(String[] args) {
     int x = 10;
   }

   public void method1(){
     int x = 20;
   }

   //local variables can not have 'access modifier' or 'static' keyword
   public void method2(){
     public int x = 20;     //!!compile error!!
     protected int y = 20;  //!!compile error!!
     private int z = 20;    //!!compile error!!

     static int a = 20;     //!!compile error!!
   }

   //Initialization is 'mandatory' for local variables
   public void method3(){
      int x;                //!!compile error!!
      System.out.println(x);
   }

}
```

▼ 2-Istance Variable:

- A variable declared "inside a class but outside a method / constructor / block"
- Belongs to **objects**, not class.
- Each object can have its own value
- Can only be accessed by creating object
- It can not be "static".
- Initialization is not mandatory for instance variables

```
public class MyClass2 {

  int x = 10;     //instance variable
  int y;          //Initialization is not mandatory for instance variables

  public static void main(String[] args) {
    //some code
  }

  public void method1(){
    //some code
  }

}

//----------------------------------------

class Test{

  public static void main(String[] args) {

    //Instance variables can only be accessed by creating object
    MyClass2 object = new MyClass2();

    System.out.println( object.x );   //10
    System.out.println( object.y );   //0
  }
}
```

▼ 3-Static Variable:

- A variable declared as static (inside a class but outside a method / constructor / block)
- Belongs to **class**, not object (class level variable)
- A single copy of variable is created and shared among all objects at class level
- Objects don't have their own values, instead the same value is shared among all the objects
- If we change the value of a static variable, all other objects will be affected by the change

- Can only be accessed by two ways:
  - a) Creating an object *(objectName.variableName)*
  - b) Directly with Class Name *(className.variableName)*

    (Prefer to access with class name)
- Initialization is not mandatory for static variables

```
public class MyClass3 {

  static int x = 10;   //static variable (class-level variable)
  static int y;    //Initialization is not mandatory for static variables

  public static void main(String[] args) {
    //some code
  }

  public void method1(){
    //some code
  }

}

//------------------------------------------------------------------------------

class Test2{

  public static void main(String[] args) {

    //How to access? 1-Creating an object
    MyClass3 object = new MyClass3();

    System.out.println( object.x );
    System.out.println( object.y );

    //How to access? 2-Directly with className
    System.out.println( MyClass3.x );
    System.out.println( MyClass3.y );

  }

}
```

▼ **Differences Between Instance and Static Variables**

```
public class MyClass {

  int x;   //instance variable

}

//---------------------------------------

class Test{

  public static void main(String[] args) {

     //1-Create objects from the class (instantiate)
     MyClass object1 = new MyClass();
     MyClass object2 = new MyClass();
     MyClass object3 = new MyClass();

     //2-assign different values to instance variable x (initialize)
     object1.x = 10;
     object2.x = 20;
     object3.x = 30;

     //3-print out each variable x
     System.out.println( object1.x  );   //10
     System.out.println( object2.x  );   //20
     System.out.println( object3.x  );   //30

}
```

```
public class MyClass {

  static int y;   //static variable

}

//---------------------------------------

class Test{

  public static void main(String[] args) {

     //1-Create objects from the class (instantiate)
     MyClass object1 = new MyClass();
     MyClass object2 = new MyClass();
     MyClass object3 = new MyClass();

     //2-assign different values to static variable y (initialize)
     object1.y = 10;
     object2.y = 20;
     object3.y = 30;
```

```
      //3-print out each variable y
      System.out.println( object1.y  );   //30
      System.out.println( object2.y  );   //30
      System.out.println( object3.y  );   //30

      //4-Or assign the value directly with class name
      MyClass.y = 50;

      System.out.println( object1.y  );   //50
      System.out.println( object2.y  );   //50
      System.out.println( object3.y  );   //50
}
```

B) Methods:

▼ B) Methods:

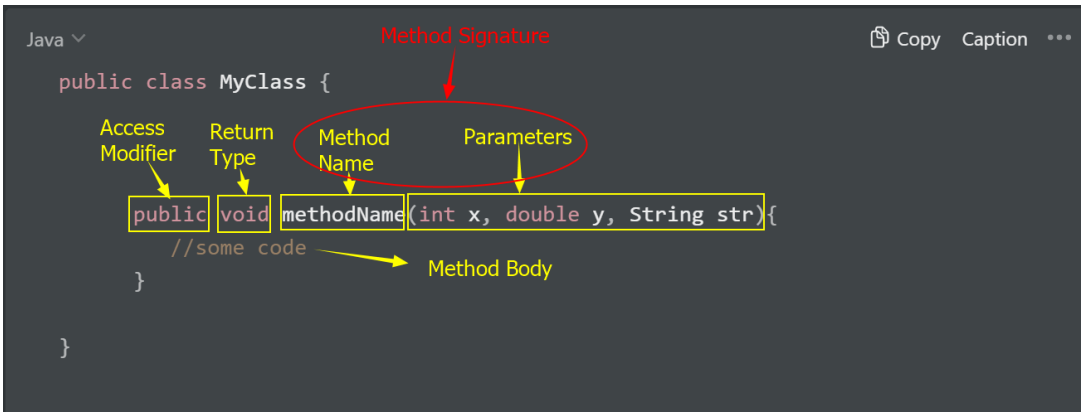
- Method: A collection of instructions / block of code to performs a specific task
- Benefit: Reusability of code
- It consist of:
  - Access Modifier
  - Return Type
  - Method Name
  - Parameters
  - Method Body *(All the code inside the curly braces)*

```
public class MyClass {

      public void methodName(int x, double y, String str){
         //some code
      }

}
```

▼ click



- In java, there are two kind of basic Methods
  - a) Instance Method
  - b) Static Method

▼ 1- Instance Method

- Method that belongs to "Object" rather than class
- Declared without "static" keyword *(Non-static method)*
- In order to call instance methods, we need to create an "Object"

```
//how to call an instance method?
public class MyClass {

   //instance method
   public void method1(){
      //some code
   }

}

//-------------------------------------------------------

class Test{

   public static void main(String[] args) {

      //Create an object in order to call instance method
      MyClass object = new MyClass();
      object.method1();

   }
```

```
}
```

▼ 2-Static Method

- Method that belongs to "Class" rather than object
- Declared with "static" keyword
- In order to call static methods, we can either:
  - Create an object: (objectName . methodName)
  - Or directly use the Class name (ClassName . methodName)

    (Prefer to call through Class Name)
- "this" keyword is can not be used in static methods

```
//how to call a static method?
public class MyClass {

  public static void method2(){                  //static method
    //some code
  }

}

//-----------------------------------------------------

class Test{

  public static void main(String[] args) {

    //either create an object
    MyClass object = new MyClass();
    object.method2();

    //Or call directly by class name
    MyClass.method2();

  }

}
```

▼ **Access Rules**

1-Instance methods can access to both instance variables/methods + static variable/methods

2-Static methods can only access to static variables/methods

Instance ⟶ instance + static
Static ⟶ only static

```
// access to instance and static variables
public class MyClass{

    public int x = 10;                      //instance varaible
    public static int y = 20;               //static varaible

    //-------------------------------------------------------------------------

    public void instanceMethod1(){          //from instance to instance ✔
       System.out.println( x );
    }

    public void instanceMethod2(){          //from instance to static ✔
       System.out.println( y );
    }

    //-------------------------------------------------------------------------

    public static void staticMethod(){            //from static to instance ✘
       System.out.println( x );                   //!!COMPILE ERROR!!
    }

    public static void staticMethod2(int x){      //from static to instance ✘
       this.x = x;                                //!!COMPILE ERROR!!
    }

    public static void staticMethod3(){           //from static to static ✔
      System.out.println( y );
    }

    //-------------------------------------------------------------------------

    //main method
    public static void main(String[] args) {
```

```
        System.out.println( x );         //from static to instance ✘

        MyClass object = new MyClass();
        System.out.println( object.x );

        System.out.println( y );         //from static to static ✔
    }

}
```

```
// access to instance and static methods
public class MyClass {

    public void instMethod(){               //instance method
        //some code
    }

    public static void staticMethod(){      //static method
        //some code
    }

    //-----------------------------------------------------------------------

    public void instanceCaller1(){          //from instance to instance ✔
        instMethod();
    }

    public void instanceCaller2(){          //from instance to static ✔
        staticMethod();
    }

    //-----------------------------------------------------------------------

    public static void staticCaller1(){     //from static to instance ✘
        instMethod();                       //!!COMPILE ERROR!!
    }

    public static void staticCaller2(){     //from static to static ✔
        staticMethod();
    }

    //-----------------------------------------------------------------------

    //main method
    public static void main(String[] args) {
        instMethod();                   //from static to instance ✘
        staticMethod();                 //from static to static ✔
    }

}
```

▼ Method Overloading

- Method overloading: Having multiple methods with the same name but with different parameters
- Benefit: to increase the readability of the program
- How to overload a method? By changing:
  - 1-Number of Parameters
  - 2-Data types of parameters
  - 3-Sequence of data types

Changing Number of Parameters

```
public class MyClass {


  public void add(int x){                       //method with one int
     //some code
  }

  public void add(int x, int y){                //method with two int
     //some code
  }

  public void add(int x, int y, int z){         //method with three int
     //some code
  }

  public void add(int...x){          //you can write as many integer as you want
     //some code
  }

}
```

Changing Data Types of Parameters

```
public class MyClass {
```

```
  public void add(int x, int y){                    //method with two int
     //some code
  }

  public void add(int x, double y){                 //method with int and double
     //some code
  }

  public void add(double x, double y){              //method with two double
     //some code
  }

}
```

Changing Sequence of Data Types

```
public class MyClass {

  public void add(int x, double y){                 //int, double
     //some code
  }

  public void add(double x, int y){                 //double, int
     //some code
  }

}
```

- Just changing "return type" does not overload a method

```
public class MyClass {

  //---!!! COMPILE ERROR !!!---
  public void add(int x, int y){                 //method with 'void' return type
     //some code
  }

  public int add(int x, int y){                 //method with 'int' return type
     //some code
  }

}
```

- Can we overload the main method?
  - Yes we can. In this situation, the method with "public static void main(String[] args)" will be executed
- Can we overload a static method?
  - Yes we can. (Remember that main method is also a static method)

Type Promotion in Java

- 1- In method overloading, if there is no exact matching data types, one data type can promote to another data type.

```
public class MyClass{

   public void add(int x){                  //method with one int
     System.out.println(x);
     System.out.println("integer");
   }

   public void add(int x, long y){          //method with int and long
     System.out.println(x + y);
     System.out.println("integer and long");
   }
}

//---------------------------------------------------------------------------

class Test{

    public static void main(String[] args) {

      int x = 10, y = 20;

      MyClass object = new MyClass();
      object.add(x,y);

      //result is:
      //30
      //integer and long

      //the argument "int y" is promoted to long data type
    }
}
```

- 2- If there is exact matching data types, no promotion will happen

```
public class MyClass {

  public void add(int x, int y){                  //method with two int
    System.out.println(x + y);
    System.out.println("integer and integer");
  }

  public void add(int x, long y){                 //method with int and long
    System.out.println(x + y);
    System.out.println("integer and long");
  }
}

//-------------------------------------------------------------------------

class Test{

  public static void main(String[] args) {

    int x = 10, y = 20;

    MyClass object = new MyClass();
    object.add(x,y);

      //result is:
      //30
      //integer and integer

      //No promotion happened because there is exact matching data types

  }
}
```

- 3- If there is no exact matching data type,
- And if more than one method is eligible for type promotion, it will give an compile error

```
public class MyClass {

  public void add(int x, long y){                  //method with int and long
    System.out.println(x + y);
```

```
    System.out.println("integer and long");
  }

  public void add(long x, int y){                  //method with long and int
    System.out.println(x + y);
    System.out.println("long and integer");
  }
}

//-------------------------------------------------------------------------

class Test{

  public static void main(String[] args) {

    int x = 10, y = 20;

    MyClass object = new MyClass();
    object.add(x,y);                       //!!! COMPILE ERROR !!!

    //we called the method with two int argument,
    //but there is no exact matching method
    //And both of these methods are eligible for type promotion
    //At this point, JVM doesn't know which method to be executed
    //So it gives a compile error!!

  }
}
```

C) Blocks

▼ C) Blocks

- A Block in java is a group of statements / code placed inside curly braces
- There are two kind of block in java:
  - Static (Initialization) Block
  - Instance (Initialization) Block

▼ Static Initialization Block

- It is used to initialize class variables (static variables)
- It is automatically called when the class is loaded to the memory

(Simply when we run the program)

- Executed before instance blocks and constructor (executed first)
- Executed only once!
- We may have more than one static init. block
- And they are executed in the order based on the source code

```
public class MyClass {

   //static variables
   public static int x;
   public static int y;

   //first static block
   static {
     x = 10;
     y = 20;

     System.out.println(x);
     System.out.println(y);
     System.out.println("first static block is executed");
   }

   //main method
   public static void main(String[] args) {
     System.out.println("main method is executed");
   }

   //second static block
   static {
     System.out.println("second static block is executed");
   }
}

//----------------------------------------------------------------------
//Output is:
  //10
  //20
  //first static block is executed
  //second static block is executed
  //main method is executed
```

- Static blocks can only access to other static members (variables, methods)

```
public class MyClass {

   public static int x;                      //static variable
   public int y;                             //instance variable

   public static void staticMethod(){        //static method
     //some code
   }

   public void instanceMethod(){             //instance method
     //some code
   }

   //-----------------------------------------------------------

   static {                  //from static to static ✔
     x = 10;
     staticMethod();
   }

   static {                  //from static to instance ✘
     y = 20;                 //!!!COMPILE ERROR!!!
   }

   static {                  //from static to instance ✘
     instanceMethod();       //!!!COMPILE ERROR!!!
   }

}
```

▼ Instance Initialization Block

- It is used to initialize instance variables
- It is automatically called whenever an object is created
- Executed after static blocks and before constructor
- Executed each time an object is created (unlike static blocks)
- We may have more than one instance block
- And they are executed in the order based on the source code
- Instance blocks can access to **both static and instance** members

```
public class MyClass {

    public int x;                   //instance variable
    public static int y;            //static varaible

    //static method
    public static void staticMethod(){
      System.out.println("static method is called from instance block");
    }

    //instance method
    public void instanceMethod(){
      System.out.println("instance method is called from instance block");
    }

    //constructor
    MyClass(){
       System.out.println("constructor is executed");
    }

    //instance block
    {
       System.out.println(x = 10);
       System.out.println(y);
       staticMethod();
       instanceMethod();
       System.out.println("instance block is executed");
    }

    //static block
    static{
       System.out.println( y = 20 );
       System.out.println("static block is executed");
    }
}

//-------------------------------------------------------

class Test{
    public static void main(String[] args) {
       MyClass object1 = new MyClass();
    }
}

//---------------------------------------------------------------------
//Output is:
  //20
  //static block is executed
  //10
  //20
  //static method is called from instance block
  //instance method is called from instance block
```

```
  //instance block is executed
  //constructor is executed
```

```
public class MyClass {

  //constructor
  MyClass(){
    System.out.println("constructor is executed");
  }

  //static block
  static{
    System.out.println("static block is executed");
  }

  //instance block
  {
    System.out.println("instance block is executed");
  }

}

//-------------------------------------------------------

class Test{
  public static void main(String[] args) {

    MyClass object1 = new MyClass();
    MyClass object2 = new MyClass();
    MyClass object3 = new MyClass();

  }
}

//-------------------------------------------------------
//Output is:
  //static block is executed
  //instance block is executed
  //constructor is executed

  //instance block is executed
  //constructor is executed

  //instance block is executed
  //constructor is executed
```

- Execution Flow:
    - 1-Static Blocks (only once)
    - 2-Instance Blocks (whenever we create an object)
    - 3-Constructors (whenever we create an object)
    - 4-Main Method