Static, this & final Keyword in Java

Dr. Partha Pratim Sarangi
School of Computer Engineering

Static Modifier

- Static is a keyword in Java, which is known as non-access modifier.
- Static means one copy per class.
- It is specific to Class and not to Instances/Objects.
- This static modifier can be used in four cases.
  - Static blocks
  - Static variables (Class variables)
  - Static methods (Class methods)
  - Static classes

Static block

- Java codes inside opening ‘{‘ and closing ‘}’ curly braces is termed as a block.
- Blocks can only be kept inside the class template.
- We can create two types of block in Java program such as Non-static and Static blocks.
- A keyword static is preceded before a block.
- Static block is executed only once.

Syntax is

```
{
    Java codes;
}
```

```
Static
{
    Java codes;
}
```

Contd…

- A class may have many static blocks and they may be appear anywhere in the class body.
- The static blocks are called in the same order as they appear in the source code.
- The nature of the static block is that the codes inside it get executed before the execution of main() method starts.
- Then, **is it possible to execute a Java program without the main method**? **Answer is Yes for java version before JDK 7!!**

```
class WithoutMain
{
        static
        {
                System.out.println("Program execution without main function");
                System.exit(0);
        }

}
```

- When **System.exit(0)** is called inside the static block, it forces the termination of the program.

Non-static block

- A block without static modifier is known as non-static block.
- The statements inside the non-static block get executed whenever an object of that class is created.
- During instantiation of a class, first the non-static blocks are executed before the execution of the codes inside the constructors of the class.
- Every time when an object of that class is created, the non-static blocks are again executed.
- It can initialize instance variables in a class.
- It initializes instance variables in an anonymous inner class which the constructor cannot initialize.

Static variables

- A static variable is class variable, but it is common to all instances of the class.
- This variable can be directly accessed and altered through class name, object name and directly accessed within a non-static method and constructor.
- Static variables are declared outside the method and constructor.
- Static variable is initialized at class loading time.
- There is only one copy of the static variable for a particular class.
- Along with public and final modifiers, static variable is used as constant value.

Static methods

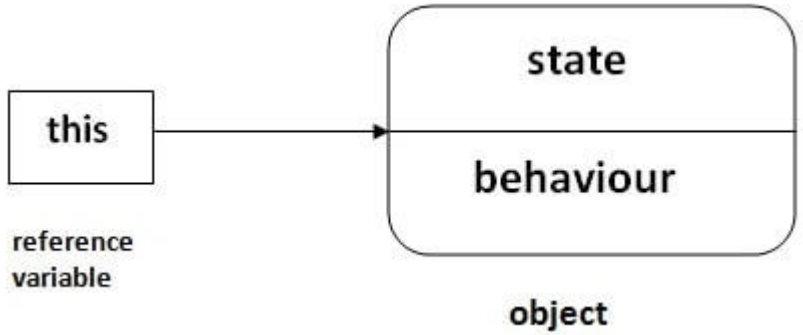
- Like static variables, static methods are tied to the class and not to the instance of the class.
- Static methods are called directly or by using class name.
- Static methods cannot access non-static methods or non-static variables directly rather non-static members are accessed by using the object of the class.
- Only one copy of the static methods are created when we load class at run time and can be accessed directly by both static and non-static methods.
- The JVM runs the static method first, followed by the creation of class instances or objects.
- In a static environment, this and super aren’t allowed to be used.

Static classes

- Java does not support static modifier on normal Java class.
- Any static member must be contained inside a class.
- Then, a class can be static only when it is contained inside another class. This gives rise to the concept of inner/nested static class. Inner/nested class is defined inside another class.
- Only nested classes can have static modifier in Java.
- They cannot access by any non-static method and variables of the class.
- A static member cannot be created inside the nested class.

this keyword in Java

- In Java, this is a reference variable that refers to the current object.



- In Java, this keyword has six usage:
  - this can be used to refer current class instance variable.
  - this can be used to invoke current class method (implicitly)
  - this() can be used to invoke current class constructor.
  - this can be passed as an argument in the method call.
  - this can be passed as argument in the constructor call.
  - this can be used to return the current class instance from the method.

Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01 this can be used to refer current class instance variable.

02 this can be used to invoke current class method (implicity)

03 this() can be used to invoke current class Constructor.

04 this can be passed as an argument in the method call.

05 this can be passed as argument in the constructor call.

06 this can be used to return the current class instance from the method

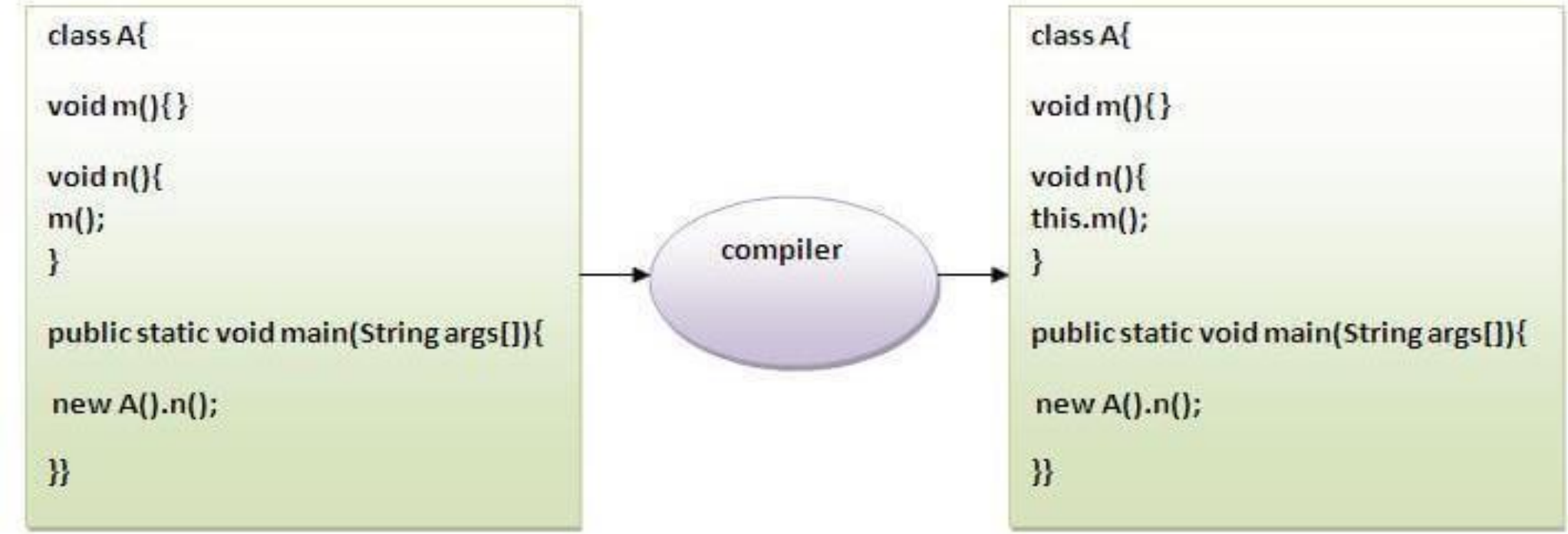
1) this: to refer current class instance variable

- The this keyword can be used to refer current class instance variable.

```
class Test
{
    int a;
    int b;

    // Parameterized constructor
    Test(int a, int b)
    {
        this.a = a;
        this.b = b;
    }

    void display()
    {
        //Displaying value of variables a and b
        System.out.println("a = " + a + "   b = " + b);
    }

    public static void main(String[] args)
    {
        Test object = new Test(10, 20);
        object.display();
    }
}
```

2) this: to invoke current class method

- To invoke the method of the current class by using the this keyword.



3) this() : to invoke current class constructor

- The this() constructor call can be used to invoke the different current class constructor. In other words, it is used for constructor chaining.

```
class Test
{
    int a;
    int b;

    //Default constructor
    Test()
    {
        this(10, 20);
        System.out.println("Inside  default constructor \n");
    }

    //Parameterized constructor
    Test(int a, int b)
    {
        this.a = a;
        this.b = b;
        System.out.println("Inside parameterized constructor");
    }

    public static void main(String[] args)
    {
        Test object = new Test();
    }
}
```

4) this: to pass as an argument in the method

- The this keyword can also be passed as an argument in the method.

```
class Test
{
    int a;
    int b;

    // Default constructor
    Test()
    {
        a = 10;
        b = 20;
    }

    // Method that receives 'this' keyword as parameter
    void display(Test obj)
    {
        System.out.println("a = " +obj.a + "  b = " + obj.b);
    }

    // Method that returns current class instance
    void get()
    {
        display(this);
    }

    public static void main(String[] args)
    {
        Test object = new Test();
        object.get();
    }
}
```

5) this: to pass as argument in the constructor call

- We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

```
class A
{
    B obj;

    // Parameterized constructor with object of B
    // as a parameter
    A(B obj)
    {
        this.obj = obj;

     // calling display method of class B
        obj.display();
    }

}
```

```
class B
{
    int x = 5;

    // Default Constructor that create a object of A
    // with passing this as an argument in the
   // constructor
    B()
    {
        A obj = new A(this);
    }

    // method to show value of x
    void display()
    {
        System.out.println("Value of x in Class B : " + x);
    }

    public static void main(String[] args) {
        B obj = new B();
    }
}
```

6) this keyword can be used to return current class instance

- We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

```
class Test
{
    int a;
    int b;

    //Default constructor
    Test()
    {
        a = 10;
        b = 20;
    }

    //Method that returns current class instance
    Test get()
    {
        return this;
    }

    //Displaying value of variables a and b
    void display()
    {
        System.out.println("a = " + a + "  b = " + b);
    }

    public static void main(String[] args)
    {
        Test object = new Test();
        object.get().display();
    }
}
```

Final keyword/Non-Access Modifier in Java