Difference between static and non static method-

TestMethod.java

```java
package javasessions;

public class TestMethod {

    public void getMail() {
        System.out.println("get mail");
    }

    public static void sendMail() {
        System.out.println("send mail");
    }

    public static void main(String[] args) {

        //how to call static methods:
        //1. using the class name:
        TestMethod.sendMail();
        //2. within the same class, use it directly:
        sendMail();


        //how to call non static methods: we need to create the object:


        TestMethod obj = new TestMethod();
        obj.getMail();
```

```
<terminated> TestMethod (1) [Jav
send mail
send mail
get mail
```

static1class

static1.java

```java
package com.day14;

public class static1 {

    public void getemail() {
        System.out.println("getting email");
    }

     public static void staticMethod() {
            System.out.println("This is a static method");
        }

     public static void main(String[] args) {

         static1 obj = new static1();

         //to call static method use classname.
         static1.staticMethod();

         //if in same class, then no need of classname also.
         staticMethod();

         //call static method with object name.
         obj.staticMethod();
         //warning - The static method staticMethod() from the type static1 should be accessed in a static way

         //to call non static method use object.
         obj.getemail();
    }

}

//This is a static method
//This is a static method
//This is a static method
//getting email
```

We get warning when accessing static with object-

```java
        obj.sendMail();

    }

}
```

The static method sendMail() from the type TestMethod should be accessed in a static way

4 quick fixes available:

- Change access to static using 'TestMethod' (declaring type)
- Remove 'static' modifier of 'sendMail()'
- Add @SuppressWarnings 'static-access' to 'main()'
- Configure problem severity

Overloading allowed for static also-

```java

    public static void sendMail() {
        System.out.println("send mail");
    }

    public static void sendMail(int a) {
        System.out.println("send mail");
    }

    public static void sendMail(int a , int b) {
        System.out.println("send mail");
    }

    public void sendMail(String a) {
        System.out.println("send mail");
    }

```

Can mix static and non static.

paste static2

static2.java

```java
package com.day14;

public class static2 {

    //can overload static no issues.
    //can mix static and non static in overload.

    public static void myMethod() {
        System.out.println("This is a static method with no parameters");
    }

    public static void myMethod(int x) {
        System.out.println("This is a static method with one integer parameter: " + x);
    }

    public void myMethod(String s) {
        System.out.println("This is a static method with one string parameter: " + s);
    }

    public static void myMethod(int x, int y) {
        System.out.println("This is a static method with two integer parameters: " + x + " and " + y);
    }


    //below method will give duplicate with first one.
//    Duplicate method myMethod() in type static2
//    public void myMethod() {
//        System.out.println("This is a non-static method with no parameters");
//    }
}

//This is a static method
//This is a static method
//This is a static method
//getting email


```

Main method overloading-

Og main method is called.

```java
package javasessions;

public class MainMethodOverloading {

    // JVM -- PSVM(String [])
    public static void main(String[] args) {

        System.out.println("hello");
    }

    public static void main(int a) {

        System.out.println("hii");
    }

}
```

```java
    public static void main(int a, int b) {

        System.out.println("bye");
    }

}
```

```
<terminated> MainMetho
hello
```

paste static4

static4.java

```java
package com.day14;

public class static4 {

    //can overload static no issues.
    //can mix static and non static in overload.
    //return type has not significance.

    public static void myMethod() {
        System.out.println("This is a static method with no parameters");
    }

    public static int myMethod(int x) {
        System.out.println("This is a static method with one integer parameter: " + x);
        return x;
    }

    public void myMethod(String s) {
        System.out.println("This is a static method with one string parameter: " + s);
    }

    public double myMethod(double s) {
        System.out.println("This is a static method with one double parameter: " + s);
        return s;
    }

    public static void myMethod(int x, int y) {
        System.out.println("This is a static method with two integer parameters: " + x + " and " + y);
    }
```

```java
    }

    public static void main(String[] args) {

        static4 s1 = new static4();

        static4.myMethod();
        static4.myMethod(10);
        s1.myMethod("Hello");
        s1.myMethod(10.0);

    }
}

//This is a static method with no parameters
//This is a static method with one integer parameter: 10
//This is a static method with one string parameter: Hello
//This is a static method with one double parameter: 10.0
//
```

//overloaded main calling another main.

TestMethod.java MainMethodOverloading.java

```java
package javasessions;

public class MainMethodOverloading {

    // JVM -- PSVM(String [])
    public static void main(String[] args) {

        System.out.println("hello");

        MainMethodOverloading.main(10);
    }

    public static void main(int a) {

        System.out.println("hii " + a);
    }

    public static void main(int a, int b) {

        System.out.println("bye");
    }

}
```

<terminated> MainMeth

```
hello
hii 10
```

```java
    public static void main(int a) {

        System.out.println("hii " + a);

        MainMethodOverloading.main(a, 30);
    }

    public static void main(int a, int b) {

        System.out.println("bye " + (a+b));
    }

}
```

```
hii 10
bye 40
```

Why main method is static-

Because jvm is calling main method.

Why main is void –

Never returns anything as no business logic.

The parameter name can be anything in main method-

Psvm(string karan[]);

```
25
26
27    public static void main(String a[]) {
28
29        System.out.println(a.length);
30
```

Length is zero as jvm not supplying anything.

paste mainmethooverload4

mainmethodoverload4.java

```java
package com.day14;

public class mainmethodoverload4 {

    public static void main(String[] args) {

        System.out.println("hello");
        mainmethodoverload4.main(10);

}

    // Overloaded main #1
    public static void main(int a) {
        System.out.println("Overloaded main with int: " + a);

    }

    public static void main(String[] arr1, String[] arr2) {
        System.out.println("Karan");
    }
}

//hello
//Overloaded main with int: 10
//only the major main is given importance in overloaded main concept.
```

Method chaining-

TestMethod.java | MainMethodOverloading.java | MethodChaining.java

```java
package javasessions;

public class MethodChaining {

    public void m1() {
        System.out.println("m1 method");
    }

    public void m2() {
        System.out.println("m2 method");
    }

    public void m3() {
        System.out.println("m3 method");
    }

    public static void t1() {
        System.out.println("t1 method");
    }

    public static void t2() {
        System.out.println("t2 method");
    }

    public static void t3() {
        System.out.println("t3 method");
    }

```

TestMethod.java | MainMethodOverloading.java | MethodChaining.java

```java
package javasessions;

public class MethodChaining {

    public void m1() {
        System.out.println("m1 method");
        m2();
    }

    public void m2() {
        System.out.println("m2 method");
        m3();
    }

    public void m3() {
        System.out.println("m3 method");
    }

    public static void t1() {
        System.out.println("t1 method");
    }

    public static void t2() {
        System.out.println("t2 method");
    }

    public static void t3() {
        System.out.println("t3 method");
    }

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

        MethodChaining obj = new MethodChaining();
        obj.m1();

```

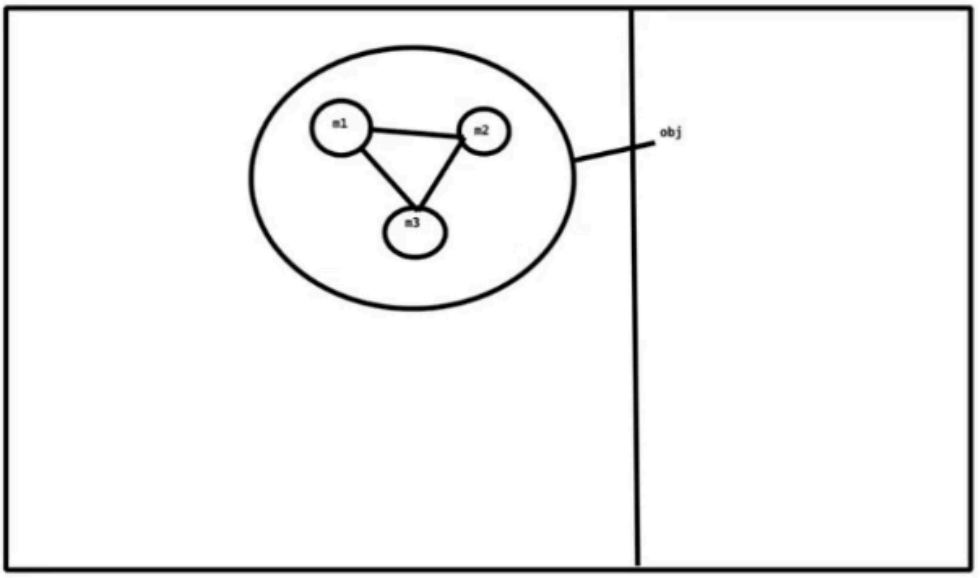
```
<terminated> MethodChaining
m1 method
m2 method
m3 method
```

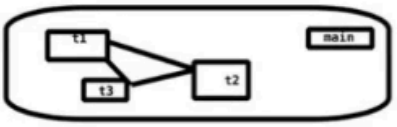
```java

    public void m3() {
        System.out.println("m3 method");
    }

    public static void t1() {
        System.out.println("t1 method");
        t2();
    }

    public static void t2() {
        System.out.println("t2 method");
        t3();
    }

    public static void t3() {
        System.out.println("t3 method");
    }

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

        MethodChaining obj = new MethodChaining();
        obj.m1();

        //
        MethodChaining.t1();

    }

```

```
<terminated> MethodChai
m1 method
m2 method
m3 method
t1 method
t2 method
t3 method
```

When in same block/class, then no need to create objects-

even if we create object no issues.


n1
n2
n3
obj


t1
t3
t2
main

TestMethod.java | MainMethodOverloading.java | MethodChaining.java

```java
    }

    public void m2() {
        System.out.println("m2 method");
        m3();
    }

    public void m3() {
        System.out.println("m3 method");
        MethodChaining.t1();
    }

    public static void t1() {
        System.out.println("t1 method");
        t2();
    }

    public static void t2() {
        System.out.println("t2 method");
        t3();
    }

    public static void t3() {
        System.out.println("t3 method");
    }

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

        MethodChaining obj = new MethodChaining();
        obj.m1();

        //
        MethodChaining.t1();

    }
```

```
<terminated> MethodChainin
m1 method
m2 method
m3 method
t1 method
t2 method
t3 method
t1 method
t2 method
t3 method
```

This is also ok-

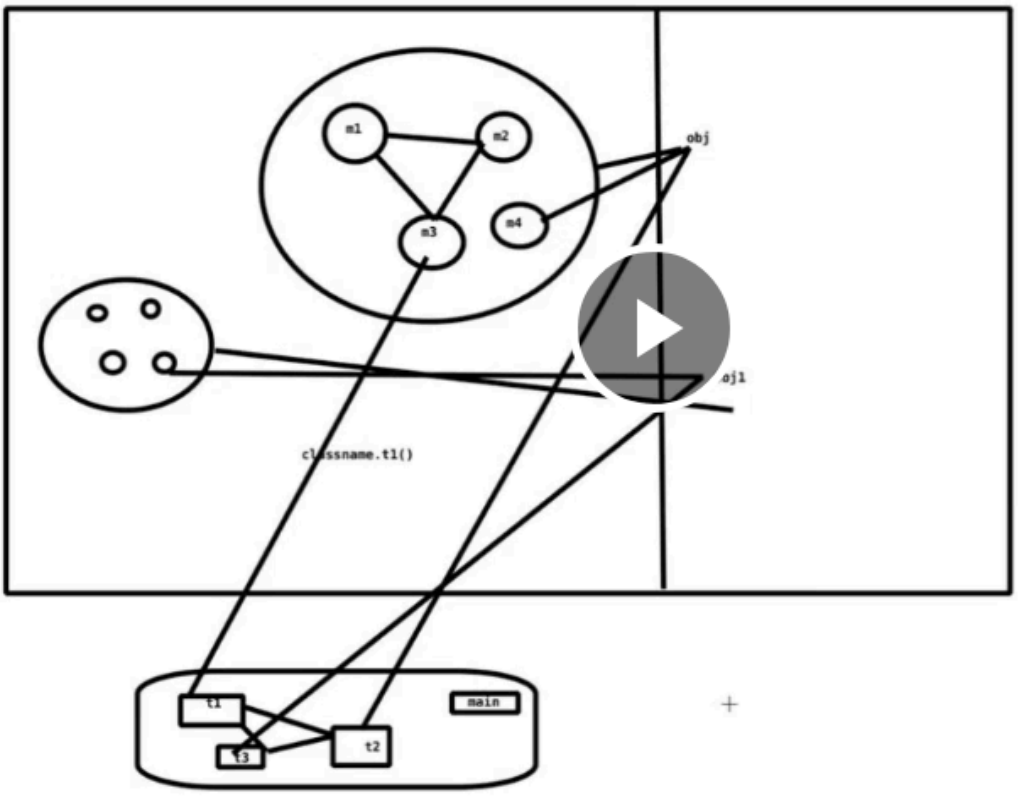
```
13      }
14
15      public void m3() {
16          System.out.println("m3 method");
17          MethodChaining.t1();
18      }
19
20      public static void t1() {
21          System.out.println("t1 method");
22          MethodChaining.t2();
23      }
24
25      public static void t2() {
26          System.out.println("t2 method");
27          MethodChaining.t3();
28      }
```

runs properly as earlier code.

Call non static from static-

```
19
20      public void m4() {
21          System.out.println("bye m4");
22      }
23
24      public static void t1() {
25          System.out.println("t1 method");
26          MethodChaining.t2();
27      }
28
29      public static void t2() {
30          System.out.println("t2 method");
31          MethodChaining.t3();
32      }
33
34      public static void t3() {
35          System.out.println("t3 method");
36          MethodChaining obj1 = new MethodChaining();
37          obj1.m4();
38          |
39      }
40
41      //JVM
```

runs properly as earlier code.




```
//NS -- NS : direct calling (no object required)
//S -- S : direct/using class name
//NS -- S: direct/using class name
//S -- NS: Create the object and then call it
```

Call by ref-

TestMethod.java | MainMethodOverloading.java | MethodChaining.java | CallByRef.

```java
package javasessions;

public class CallByRef {


    public void m1(int a) {
        System.out.println(a);
    }



    public static void main(String[] args) {

        CallByRef obj = new CallByRef();
        obj.m1(20);//call by value/argument

```

```
<terminated> CallB
20
```

Explanation-

Clipboard | Image | Tools | Shapes | Colors

```java
public class CallByRef {

    String name;
    int age;

    public void m1(int a) {
        System.out.println(a);
    }

    public static void t1(CallByRef p1) {
        p1.m1();
    }

    public static void main(String[] args) {

        CallByRef obj = new CallByRef();
        obj.m1(20);// call by value/argument

        CallByRef.t1(obj);
    }

}
```

p1=obj

call by ref

name
age
m1()

obj

t1()

TestMethod.java | MainMethodOverloading.java | MethodChaining.java | CallByRef.ja

```java
package javasessions;

public class CallByRef {

    String name;
    int age;

    public void m1(int a) {
        System.out.println(a);
    }

    public static void t1(CallByRef p1) {
        p1.m1(20);
    }

    public static void main(String[] args) {

        CallByRef obj = new CallByRef();
        obj.m1(20);// call by value/argument

        CallByRef.t1(obj);

    }

}
```

```
<terminated> CallE
20
20
```

We can only pass object when calling t1 not any other data type-

paste callby3

callby3.java

```java
package com.day14;

public class callby3 {

public void m1(int a) {
    System.out.println(a);
}

public static void t1(callby3 p1) {
    p1.m1(40);
}

public static void main(String[] args) {
    callby3 c1=new callby3();
    c1.m1(10); //call by value or argument.

    //in call by reference we can only pass the reference or object not primitive.
    callby3.t1(100); //The method t1(callby3) in the type callby3 is not applicable for the arguments (int)
}


}
```

TestMethod.java | MainMethodOverloading.java | MethodChaining.java | CallByRef.java

```java
package javasessions;

public class CallByRef {

    String name;
    int age;

    public void m1(int a) {
        System.out.println(a);
    }

    public static void t1(CallByRef p1) {
        p1.m1(20);
        System.out.println(p1.name);
        System.out.println(p1.age);
    }

    public static void main(String[] args) {

        CallByRef obj = new CallByRef();

        obj.name = "Naveen";
        obj.age = 20;


        //obj.m1(20);// call by value/argument

        CallByRef.t1(obj);//call by reference

    }

}
```

Naveen

20

```java
package javasessions;

public class CallByRef {

    String name;
    int age;

    public void m1(int a) {
        System.out.println(a);
    }

    public static void t1(CallByRef p1) {
        p1.m1(20);
        System.out.println(p1.name);
        System.out.println(p1.age);

        p1.name = "Tom";
        p1.age = 40;
    }

    public static void main(String[] args) {

        CallByRef obj = new CallByRef();

        obj.name = "Naveen";
        obj.age = 20;


        //obj.m1(20);// call by value/argument

        CallByRef.t1(obj);//call by reference


        System.out.println(obj.name);
        System.out.println(obj.age);

    }
```

20

naveen

20

Tom

40

TestMethod.java | MainMethodOverloading.java | MethodChaining.java | CallByRef.java

```java
1  package javasessions;
2
3  public class CallByRef {
4
5      String name;
6      int age;
7
8      public void m1(int a) {
9          System.out.println(a);
10     }
11
12     public static void t1(CallByRef p1) {
13         p1.m1(20);
14         System.out.println(p1.name);//naveen
15         System.out.println(p1.age);//20
16
17         p1.name = "Tom";
18         p1.age = 40;
19     }
20
21     public static void main(String[] args) {
22
23         CallByRef obj = new CallByRef();
24
25         System.out.println(obj.name);//null
26         System.out.println(obj.age);//0
27
28         obj.name = "Naveen";
29         obj.age = 20;
30
31
32         //obj.m1(20);// call by value/argument
33
34         CallByRef.t1(obj);//call by reference
35
36
37         System.out.println(obj.name);//tom
38         System.out.println(obj.age);//40
39
40     }
41
42 }
```

```
<terminated> CallByRef (2) [Java Applicati
null
0
20
Naveen
20
Tom
40
```

Call by reference helps avoid unnecessary object creations.

When method overloading wont work-

because all parameters maybe of one string type.

```
    }

    public void payment(String upiOrPayPalId) {

        if(upiOrPayPalId.contains("upi")) {
            //use gPay
        }
        else {
            System.out.println("paypal");
        }
```

We can have one method and use if else condition checks.

Same method written another way-

```java
31
32    public static void payment(String upiOrPayPalId, String paymentType) {
33
34        if(paymentType.equals("upi")) {
35
36        }
37        else if(paymentType.equals("paypal")) {
38
39        }
```

```java
1  package javasessions;
2
3  public class MainMethodOverloading {
4
5      // JVM -- PSVM(String [])
6      public static void main(String test[]) {
7
8          System.out.println("hello");
9
10         MainMethodOverloading.main(10);
11
12
13         payment("naveen@okhdfc", "upi");
14     }
15
```

paste payment1 for reference -

payment1.java

```java
package com.day14;

public class payment1 {

    public void payment(String upiid) {
        if((upiid).equalsIgnoreCase("card")){
            System.out.println("use your cc");
        }
        else {
            System.out.println("use your upi");
        }
    }

    public static void payment(String upiid,int amount) {
        if((upiid).equalsIgnoreCase("card")){
            System.out.println("use your cc");
        }
        else if ((upiid).equalsIgnoreCase("cash")) {
            System.out.println("pay by cash");
        }
    }

    public static void main(String[] args) {
        payment1 p1=new payment1();
        payment1.payment("cash", 3234);

    }

}

//pay by cash
```

This is also overloading-

```java
    public void m1(int a, String b) {
        System.out.println(a);
    }


    m1(string, int)
```

Changing parameters.

paste overload1

overload1.java

```java
package com.day14;

public class overload1 {

    public void m1(int a , String b) {
        System.out.println(a+b);
    }

    //overload method.
    //return type doesnt matter.
    public int m1(String a, int b) {
        System.out.println(a);
        return 1;
    }

    //overload method.
        //return type doesnt matter.
    public double m1(int b) {
        System.out.println(b);
        return 10.78;
    }

    //below will give error.
    //Duplicate method m1(int) in type overload1
//  private void m1(int a) {
//      System.out.println(a);
//  }

    //even public private doesnt matter for overloading.
    //but warning for private as not used anywhere.
    //The method m1(int, String, char) from the type overload1 is never used locally.
    private void m1(int test, String test1, char test2) {
    System.out.println(test+test1+test2);
```

```java
    private void m1(int test, String test1, char test2) {
    System.out.println(test+test1+test2);
}

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

paste overload 2

overload2.java

```java
package com.day14;

public class overload2 {

    public void m1(int a , String b) {
        System.out.println(a+b);
    }

    //overload method.
    //return type doesnt matter.
    public int m1(String a, int b) {
        System.out.println(a);
        return 1;
    }

    //overload method.
        //return type doesnt matter.
    public double m1(int b) {
        System.out.println(b);
        return 10.78;
    }

    //below will give error.
    //Duplicate method m1(int) in type overload1
//  private void m1(int a) {
//      System.out.println(a);
//  }

    //even public private doesnt matter for overloading.
    private void m1(int test, String test1, char test2) {
    System.out.println(test+test1+test2);
}
```

```java
    System.out.println(test+test1+test2);
}

    //now the private is used so warning is gone above private.
    public void m1() {
        m1(10, "test", 'c');
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

See the code-

paste callby7

callby7.java

```java
package com.day14;

public class callby7 {

    //complex program increased by one level.

    String name;
    int age;

    public void m1(int a) {
        System.out.println(a);
    }

    public static void t1(callby7 p1) {
        p1.m1(20);
        System.out.println(p1.name);
        System.out.println(p1.age);

        p1.name="harry";
        p1.age=23434;
    }

    //another example of overload in call by reference.
    //just changed the reference name and it is overloaded.

    //remember method names cannot be same else we get error.
    //Duplicate method t1(callby7) in type callby7.
    //even for overloaded ones, hence the parameter have to be different.
    public static void t2(callby7 p2) {
        p2.m1(20);
        System.out.println(p2.name);
        System.out.println(p2.age);
```

```
        System.out.println(p2.name);
        System.out.println(p2.age);

        p2.name="sejal";
        p2.age=32423434;
    }

    public static void main(String[] args) {
        callby7 c1=new callby7();

        System.out.println("after creating the object");
        System.out.println(c1.name); //null
        System.out.println(c1.age); //0

        c1.name="tiger";
        c1.age=10;

        callby7.t1(c1); //call by reference.

        callby7.t2(c1); //call by reference.

        System.out.println("print from main method");
        System.out.println(c1.name);
        System.out.println(c1.age);
```

```
54          System.out.println(c1.age);
55
56      }
57
58  }
59
60
61  //after creating the object
62  //null
63  //0
64  //20
65  //tiger
66  //10
67  //20
68  //harry
69  //23434
70  //print from main method
71  //sejal
72  //32423434
73
```