Difference between static and non static method-

```
☑ TestMethod.java 🎛
  1 package javasessions;
  3 public class TestMethod {
  4
  5⊜
         public void getMail() {
  6
              System.out.println("get mail");
  7
  8
  9⊜
         public static void sendMail() {
              System Fout.println("send mail");
 10
 11
12
       public static void main(String[] args) {
13⊜
14
           //how to call static methods:
15
           //1. using the class name:
16
17
          TestMethod.sendMail();
18
           //2. within the same class, use it directly:
19
          sendMail();
20
21
22
           //how to call non static methods: we need to create the object:
23
24
25
           TestMethod obj = new TestMethod();
26
           obj.getMail();
  <terminated> TestMethod (1) [Jan
  send mail
  send mail
  get mail
```

static1class

```
□ □ 🔝 static1.java ×
        1 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");
         10
         11
                   public static void main(String[] args) {
         13⊜
         14
                       static1 obj = new static1();
         16
17
                      //to call static method use classname.
         18
                       static1.staticMethod();
         19
                       //if in same class, then no need of classname also.
         20
         21
                       staticMethod();
         22
         23
                       //call static method with object name.
                       obj.staticMethod();
//warning - The static method staticMethod() from the type static1 should be accessed in a static way
        <u>6</u>24
         25
         26
27
                        //to call non static method use object.
         28
                       obj.getemail();
                 }
         29
         30
         31 }
         32
         33 //This is a static method
         34 //This is a static method
35 //This is a static method
         36 //getting email
```

We get warning when accessing static with object-

```
Obj. sendMail();

The static method sendMail() from the type TestMethod should be accessed in a **_tatic 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-

```
8
90
       public static void sendMail() {
           System.out.println("send mail");
10
11
12
13⊕
       public static void sendMail(int a) {
14
           System.out.println("send mail");
15
16
17⊕
       public static void sendMail(int a , int b) {
18
           System.out.println("send mail");
19
20
21⊜
       public void sendMail(String a) {
           System.out.println("send mail");
22
23
24
```

Can mix static and non static.

paste static2

```
□ □ I static2.java ×
        1 package com.day14;
          3 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");
          80
         10
         11
         12⊖
                  public static void myMethod(int x) {
                       System.out.println("This is a static method with one integer parameter: " + x);
         13
         14
         15
                  public void myMethod(String s) {
    System.out.println("This is a static method with one string parameter: " + s);
         16⊖
         17
         18
         19
                  public static void myMethod(int x, int y) {
    System.out.println("This is a static method with two integer parameters: " + x + " and " + y);
         20⊝
         21
         22
         24
         25
                   //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");
         26 //
27 //
         28
         29
         30
             }
         31
         32 //This is a static method
         33
            //This is a static method
//This is a static method
         34
         35 //getting email
         37
```

Main method overloading-

Og main method is called.

```
1 package javasessions;
 3 public class MainMethodOverloading {
  5
        // JVM -- PSVM(String [])
        public static void main(String[] args) {
 6e
 8
            System.out.println("hello");
        }
 9
10
11∘
        public static void main(int a) {
12
13
            System.out.println("hii");
        } I
14
15
16 }
          public static void main(int a, int b) {
   16⊜
   17
               System.out.println("bye");
   18
          }
   19
   20
   21 }
   22
   23
```

```
reterminated> MainMethod
```

paste static4

```
□ □ static4.java ×
        package com.day14;
        3 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() {
        9⊝
                   System.out.println("This is a static method with no parameters");
       10
       11
       13⊖
               public static int myMethod(int x) {
                   System.out.println("This is a static method with one integer parameter: " + x);
       14
       15
                   return x;
       16
              public void myMethod(String s) {
    System.out.println("This is a static method with one string parameter: " + s);
       18⊖
       19
       20
       21
       22⊖
              public double myMethod(double s) {
       23
                   System.out.println("This is a static method with one double parameter: " + s);
       24
25
                   return s;
       26
               public static void myMethod(int x, int y) {
                   System.out.println("This is a static method with two integer parameters: " + x + " and " + y);
       29
```

```
Oyocciii Oucipi ziiczii iii zo
29
       }
30
       public static void main(String[] args) {
31⊖
32
33
            static4 s1 = new static4();
34
35
            static4.myMethod();
            static4.myMethod(10);
36
37
            s1.myMethod("Hello");
            s1.myMethod(10.0);
38
39
40
       }
41 }
42
43 //This is a static method with no parameters
44 //This is a static method with one integer parameter: 10
45 //This is a static method with one string parameter: Hello
46 //This is a static method with one double parameter: 10.0
47 //
48
```

//overloaded main calling another main.

```
☑ TestMethod.java
☑ MainMethodOverloading.java ※
  1 package javasessions;
  3 public class MainMethodOverloading {
  5
        // JVM -- PSVM(String [])
        public static void main(String[] args) {
  68
  7
  8
             System.out.println("hello");
  9
 10
            MainMethodOverloading.main(10);
        }
 11
 12
 13⊕
        public static void main(int a) {
 14
             System.out.println("hii " + a);
 15
 16
 17
        public static void main(int a, int b) {
 18=
 19
 20
             System.out.println("bye");
        }
 21
 22
 23 }
 24
```

<terminated> MainMeth
hello
hii 10

```
14
       public static void main(int a) {
13⊕
14
15
           System.out.println("hii " + a);
16
17
           MainMethodOverloading.main(a, 30);
       }
18
19
       public static void main(int a, int b) {
20⊜
21
22
           System.out.println("bye " + (a+b));
       }
23
24
25 }
26
```

```
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);
```

Length is zero as jvm not supplying anything.

paste mainmethooverload4

```
🗾 mainmethodoverload4.java 🗙
  package com.day14;
  3
    public class mainmethodoverload4 {
  4
  5⊜
        public static void main(String[] args) {
  6
  7
            System.out.println("hello");
  8
            mainmethodoverload4.main(10);
  9
 10
 11
        // Overloaded main #1
 12
 13⊜
        public static void main(int a) {
 14
            System.out.println("Overloaded main with int: " + a);
 15
 16
 17
        public static void main(String[] arr1, String[] arr2) {
 18⊖
 19
            System.out.println("Karan");
 20
 21 }
 22
 23 //hello
 24 //Overloaded main with int: 10
 25 //only the major main is given importance in overloaded main concept.
 27
 28
 29
```

Method chaining-

```
☑ TestMethod.java ☑ MainMethodOverloading.java ☑ MethodChaining.java ※

   1 package javasessions;
  3 public class MethodChaining {
  4
         public void m1() {
  58
             System.out.println("m1 method");
  6
  7
  8
         public void m2() {
  90
 10
             System.out.println("m2 method");
 11
 12
 13⊕
         public void m3() {
             System.out.println("m3 method");
 14
 15
 16
         public static void t1() {
 17⊕
 18
             System.out.println("t1 method");
 19
 20
 218
         public static void t2() {
             System.out.println("t2 method");
 22
 23
 24
 258
         public static void t3() {
 26
             System.out.println("t3 method");
 27
         }
28
```

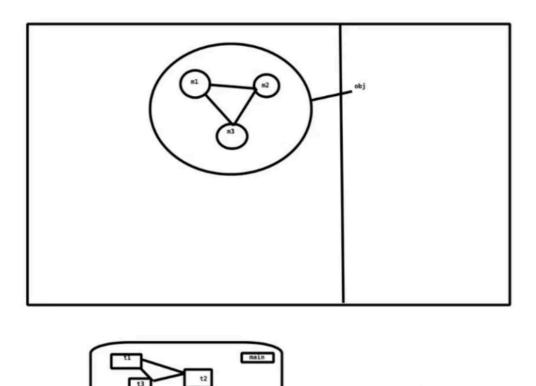
```
☐ TestMethod.java ☐ MainMethodOverloading.java ☐ MethodChaining.java ☒
 1 package javasessions;
 3 public class MethodChaining {
 5⊕
        public void m1() {
 6
            System.out.println("m1 method");
 7
            m2();
 8
        }
 9
        public void m2() {
100
            System.out.println("m2 method");
11
12
            m3();
13
        }
14
15⊕
        public void m3() {
            System.out.println("m3 method");
16
        }
17
18
19⊕
        public static void t1() {
20
            System.out.println("t1 method");
21
22
        public static void t2() {
23⊕
24
            System.out.println("t2 method");
25
26
27⊕
       public static void t3() {
28
            System.out.println("t3 method");
29
        }
30
31
        //JVM
32€
        public static void main(String[] args) {
33
            MethodChaining obj = new MethodChaining();
34
35
            obj.m1();
36
```

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

```
14
       public void m3() {
15⊕
           System.out.println("m3 method");
16
17
18
19⊕
       public static void t1() {
20
           System.out.println("t1 method");
21
           t2();
       }
22
23
       public static void t2() {
24⊖
25
           System.out.println("t2 method");
26
           t3();
       }
27
28
       public static void t3() {
29⊕
           System.out.println("t3 method");
30
31
32
33
       //JVM
       public static void main(String[] args) {
34⊕
35
36
           MethodChaining obj = new MethodChaining();
37
           obj.m1();
38
39
           MethodChaining.t1();
40
41
       }
42
43
 <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.



```
☑ TestMethod.java ☑ MainMethodOverloading.java ☑ MethodChaining.java ☑
        }
 8
 9
100
        public void m2() {
            System.out.println("m2 method");
11
12
            m3();
        }
13
14
        public void m3() {
15⊕
            System.out.println("m3 method");
16
17
            MethodChaining.t1();
18
        }
19
20⊕
        public static void t1() {
21
            System.out.println("t1 method");
22
            t2();
23
        }
24
25⊕
        public static void t2() {
26
            System.out.println("t2 method");
27
            t3();
        }
28
29
        public static void t3() {
30e
31
            System.out.println("t3 method");
32
33
34
        //JVM
        public static void main(String[] args) {
35⊕
36
37
            MethodChaining obj = new MethodChaining();
38
            obj.m1();
39
40
            //
41
            MethodChaining.t1();
42
43
        3
```

This is also ok-

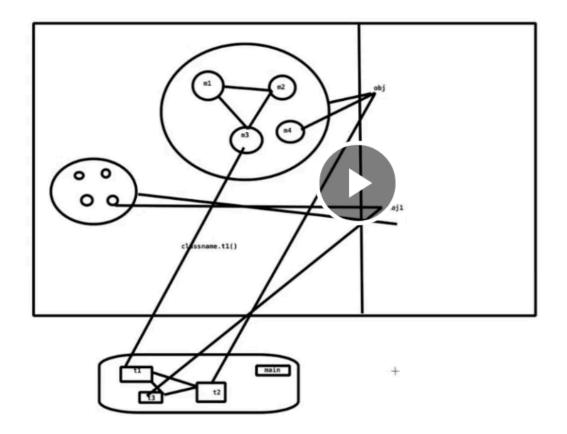
```
}
13
14
15⊖
       public void m3() {
16
           System.out.println("m3 method");
17
           MethodChaining.t1();
18
19
20⊝
       public static void t1() {
           System.out.println("t1 method");
21
22
           MethodChaining.t2();
23
       }
24
25⊜
       public static void t2() {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

runs properly as earlier code.

Call non static from static-

```
20⊖
       public void m4() {
21
           System.out.println("bye m4");
22
23
248
       public static void t1() {
           System.out.println("t1 method");
25
           MethodChaining.t2();
26
27
28
29⊜
       public static void t2() {
           System.out.println("t2 method");
30
31
           MethodChaining.t3();
32
33
34⊜
       public static void t3() {
           System.out.println("t3 method");
35
           MethodChaining obj1 = new MethodChaining();
36
37
           obj1.m4();
38
       }
39
40
41
```

runs properly as earlier code.

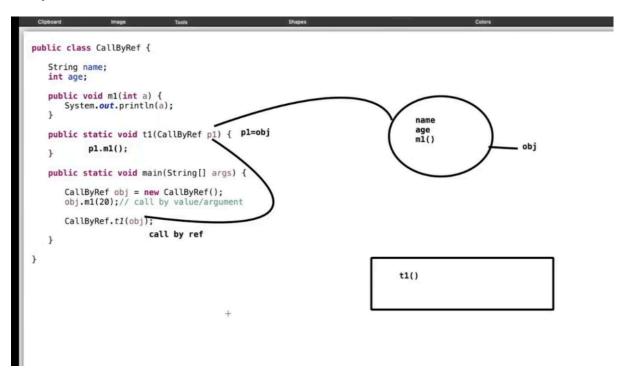


```
//NS -- NS : direct calling (no object required)
//S -- S : direct/using class name
//NS -- S: direct/using class name
//S -- NS: Create the object and then call it
```

Call by ref-

```
☐ TestMethod.java ☐ MainMethodOverloading.java ☐ MethodChaining.java ☐ CallByRef.
  1 package javasessions;
  3 public class CallByRef {
  5
  69
         public void m1(int a) {
  7
             System.out.println(a);
  8
  9
10
11
12
         public static void main(String[] args) {
13⊕
14
15
             CallByRef obj = new CallByRef();
16
             obj.m1(20);//call by value/argument
17
18
 <terminated> CallB
 20
```

Explanation-



```
☑ TestMethod.java ☑ MainMethodOverloading.java ☑ MethodChaining.java ☑ CallByRef.ja
  1 package javasessions;
 3 public class CallByRef {
 5
        String name;
 6
        int age;
 88
        public void m1(int a) {
 9
             System.out.println(a);
10
11
        public static void t1(CallByRef p1) {
12⊕
13
             p1.m1(20);
14
15
16⊜
        public static void main(String[] args) {
17
18
             CallByRef obj = new CallByRef();
19
             obj.m1(20);// call by value/argument
20
             CallByRef.t1(obj);
21
22
23
        }
24
25 }
26
 <terminated> CallE
 20
 20
```

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

paste callby3

```
callby3.java ×
       1 package com.day14;
        3 public class callby3 {
        5⊖ public void m1(int a) {
               System.out.println(a);
        9 public static void t1(callby3 p1) {
              p1.m1(40);
       11 }
       12
       13@ public static void main(String[] args) {
               callby3 c1=new callby3();
               c1.m1(10); //call by value or argument.
       15
       16
               //in call by reference we can only pass the reference or object not primitive.
      18
               callby3. \pm 1(100); //The method \pm 1(callby3) in the type callby3 is not applicable for the arguments (\underline{\text{int}})
       19 }
       20
       21
       22 }
       24
       25
       26
```

```
1 package javasessions;
12
   3 public class CallByRef {
   4
   5
         String name;
   6
         int age;
   7
   89
         public void m1(int a) {
   9
             System.out.println(a);
  10
   11
   120
         public static void t1(CallByRef p1) {
   13
             p1.m1(20);
   14
             System.out.println(p1.name);
   15
             System.out.println(pl.age);
         }
   16
   17
         public static void main(String[] args) {
   18⊕
   19
   20
             CallByRef obj = new CallByRef();
   21
   22
             obj.name = "Naveen";
   23
             obj.age = 20;
   24
   25
   26
             //obj.m1(20);// call by value/argument
   27
             CallByRef.t1(obj);//call by reference
   28
   29
         }
   30
  31
  32 }
  33
   34
```

Naveen

20

```
J MainMethodOverloading.java
                                TestMethod.java
  1 package javasessions;
  3 public class CallByRef {
  4
  5
        String name;
  6
        int age;
  7
  89
        public void m1(int a) {
  9
            System.out.println(a);
 10
 11
 12⊕
        public static void t1(CallByRef p1) {
 13
            p1.m1(20);
            System.out.println(p1.name);
 14
 15
            System.out.println(p1.age);
 16
            p1.name = "Tom";
 17
 18
            p1.age = 40;
 19
 20
        public static void main(String[] args) {
 210
 22
 23
            CallByRef obj = new CallByRef();
 24
 25
            obj.name = "Naveen";
 26
            obj.age = 20;
 27
 28
 29
            //obj.m1(20);// call by value/argument
 30
 31
            CallByRef.t1(obj);//call by reference
 32
 33
 34
            System.out.println(obj.name);
 35
            System.out.println(obj.age);
 36
```

20

naveen

20

Tom

```
☑ TestMethod.java ☑ MainMethodOverloading.java ☑ MethodChaining.java ☑ CallByRef.java
    1 package javasessions;
12
    3 public class CallByRef {
    4
    5
           String name;
    6
           int age;
    7
    89
           public void m1(int a) {
    9
               System.out.println(a);
   10
   11
           public static void t1(CallByRef p1) {
   129
   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
           public static void main(String[] args) {
   210
   22
   23
               CallByRef obj = new CallByRef();
   24
   25
               System.out.println(obj.name);//null
   26
               System.out.println(obj.age);//0
   27
               obj.name = "Naveen";
   28
   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 Applicationull
0
20
Naveen
20
Tom
40
```

Call by reference helps avoid unnecessary object creations.

When method overloading wont workbecause 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-

```
31
      public static void payment(String upiOrPayPalId, String paymentType) {
32⊕
33
          if(paymentType.equals("upi")) {
34
35
36
          else if(paymentType.equals("paypal")) {
37
38
39
          }
   1 package javasessions;
   3 public class MainMethodOverloading {
          // JVM -- PSVM(String [])
          public static void main(String test[]) {
              System.out.println("hello");
   8
   9
              MainMethodOverloading.main(10);
  10
  11
  12
              payment("naveen@okhdfc", "upi");
  13
  14
          }
```

paste payment1 for reference -

```
_ _
      🕖 payment1.java 💢
           package com.day14;
        3
           public class payment1 {
        4
        5⊜
               public void payment(String upiid) {
        6⊜
                   if((upiid).equalsIgnoreCase("card")){
        7
                        System.out.println("use your cc");
        8
                   else {
        9⊝
       10
                        System.out.println("use your upi");
       11
                   }
       12
               }
       13
               public static void payment(String upiid,int amount) {
       14⊖
       15⊜
                   if((upiid).equalsIgnoreCase("card")){
                        System.out.println("use your cc");
       16
       17
       18⊜
                   else if ((upiid).equalsIgnoreCase("cash")) {
                        System.out.println("pay by cash");
       19
       20
                   }
       21
               }
       22
       23⊜
               public static void main(String[] args) {
      24
                   payment1 p1=new payment1();
                   payment1.payment("cash", 3234);
       25
       26
       27
               }
       28
       29
       30
       31
           //pay by cash
       32
       33
```

This is also overloading-

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

m1(string, int)
```

Changing parameters.

paste overload1

```
□ □ 🕡 overload1.java 🗡
        package com.day14;
        3 public class overload1 {
        5⊝
               public void m1(int a , String b) {
                  System.out.println(a+b);
        6
        7
        8
        9
              //overload method.
               //return type doesnt matter.
       10
       11⊖
              public int m1(String a, int b) {
                   System.out.println(a);
       12
       13
                   return 1;
       14
       15
             //overload method.
       16
       17
                   //return type doesnt matter.
       18⊖
              public double m1(int b) {
       19
                  System.out.println(b);
       20
                   return 10.78;
       21
       22
              //below will give error.
       23
       24
               //Duplicate method m1(int) in type overload1
       25 // private void m1(int a) {
       26 //
27 //
                  System.out.println(a);
       29
               //even public private doesnt matter for overloading.
       30
               //but warning for private as not used anywhere.
       31
               //The method m1(int, String, char) from the type overload1 is never used locally.
               private void m1(int test, String test1, char test2) {
      32⊖
       33
               System.out.println(test+test1+test2);
```

```
// The mechod mit(inc, octing, char / from the type over toa
№32⊝
         private void m1(int test, String test1, char test2) {
         System.out.println(test+test1+test2);
 33
 34 }
 35
 36⊜
         public static void main(String[] args) {
237
             // TODO Auto-generated method stub
 38
 39
         }
 40
 41
 42
```

paste overload 2

```
_ _
      overload2.java X
          package com.day14;
        3
          public class overload2 {
        4
        5⊝
               public void m1(int a , String b) {
        6
                   System.out.println(a+b);
        7
        8
        9
               //overload method.
               //return type doesnt matter.
       10
               public int m1(String a, int b) {
       11⊖
       12
                   System.out.println(a);
                   return 1;
       13
       14
               }
       15
       16
               //overload method.
                   //return type doesnt matter.
       17
               public double m1(int b) {
       18⊖
       19
                   System.out.println(b);
       20
                   return 10.78;
       21
               }
       22
       23
               //below will give error.
       24
               //Duplicate method m1(int) in type overload1
       25 // private void m1(int a) {
       26 //
                   System.out.println(a);
       27 //
       28
       29
               //even public private doesnt matter for overloading.
       30⊝
               private void m1(int test, String test1, char test2) {
       31
               System.out.println(test+test1+test2);
       32 }
```

```
System.out.println(test+test1+test2);
 31
 32 }
33
 34
        //now the private is used so warning is gone above private.
35⊜
        public void m1() {
            m1(10, "test", 'c');
 36
 37
 38
39⊜
        public static void main(String[] args) {
240
             // TODO Auto-generated method stub
41
42
        }
43
44 }
45
```

See the code-

paste callby7

```
□ □ □ callby7.java ×
           package com.day14;
         1
           public class callby7 {
         3
         5
                //complex program increased by one level.
         6
         7
                String name;
               int age;
         8
         9
        10⊝
                public void m1(int a) {
        11
                    System.out.println(a);
        12
        13
                public static void t1(callby7 p1) {
        14⊖
        15
                    p1.m1(20);
        16
                    System.out.println(p1.name);
                    System.out.println(p1.age);
        17
        18
                    p1.name="harry";
        19
                    p1.age=23434;
        20
        21
        22
        23
                //another example of overload in call by reference.
                //just changed the reference name and it is overloaded.
        24
        25
                //remember method names cannot be same else we get error.
        26
                //Duplicate method t1(callby7) in type callby7.
        27
        28
                //even for overloaded ones, hence the parameter have to be different.
                public static void t2(callby7 p2) {
        29⊖
                    p2.m1(20);
        30
                    System.out.println(p2.name);
        31
                    System.out.println(p2.age);
        32
```

```
__..._\_-/,
           System.out.println(p2.name);
31
           System.out.println(p2.age);
32
33
34
           p2.name="sejal";
           p2.age=32423434;
35
       }
36
37
       public static void main(String[] args) {
38⊖
           callby7 c1=new callby7();
39
40
           System.out.println("after creating the object");
41
           System.out.println(c1.name); //null
42
           System.out.println(c1.age); //0
43
44
           c1.name="tiger";
45
           c1.age=10;
46
47
           callby7.t1(c1); //call by reference.
48
49
           callby7.t2(c1); //call by reference.
50
51
           System.out.println("print from main method");
52
           System.out.println(c1.name);
53
           System.out.println(c1.age);
54
55
```

```
System.out.println(c1.age);
54
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
```