

```
OCJA 1.8 Java SE 8 Programmer - I (1Z0-808) by Durga Sir On 08-04-2018
1
2
3
4 To write very concise Code
5 To enable functional programming features in java
6 -----
7 Lambda Calculas 1930
8 LISP
9 C#.Net
10 C
11 C++
12 Objective C
13 Python
14 Ruby
15 ...
16 Java also
17 |
```

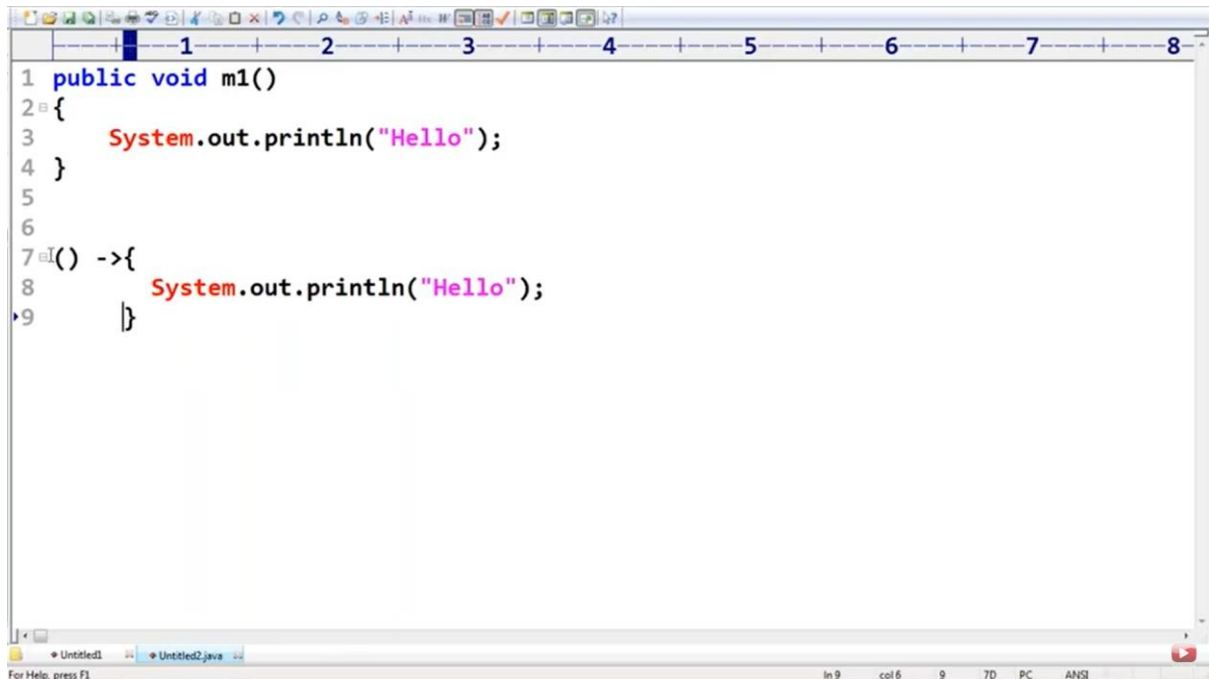
```
OCJA 1.8 Java SE 8 Programmer - I (1Z0-808) by Durga Sir On 08-04-2018
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16 Java also
17
18 What is Lambda Expression:
19 -----
20 Just an Anonymous(Nameless)FunctionI
21 No Name
22 No return type
23 No modifiers
24 |
25
26
27
28
29
30
31
32
33
34
```

Remove name

Remove return type

Remove Modifier

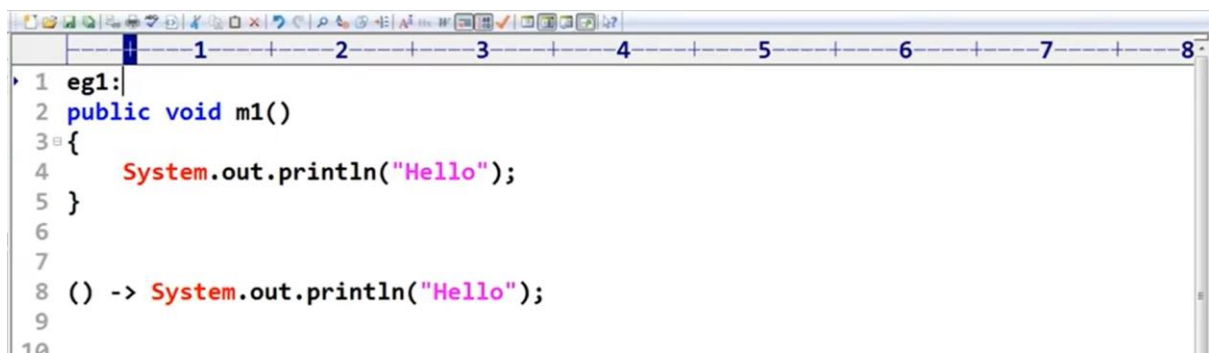
Just add an Arrow- this is anonymous function



```
1 public void m1()
2 {
3     System.out.println("Hello");
4 }
5
6
7 () ->{
8     System.out.println("Hello");
9 }
```

If its just 1 line u can remove curly braces also

()-> Sysout("Hello");



```
1 eg1:
2 public void m1()
3 {
4     System.out.println("Hello");
5 }
6
7
8 () -> System.out.println("Hello");
9
10
```

```

10
11 eg2:
12 public void add(int a,int b)
13 {
14     System.out.println(a+b);
15 }
16
17
18 (int a,int b)-> System.out.println(a+b);
19

```

Compiler can figure out the type by so we can remove it

```

18 (int a,int b)-> System.out.println(a+b);
19 (a,b)-> System.out.println(a+b);
20

```

If only single line you can remove curly brackets

Since compile figures out the type you can remove the type from arg

Since we have only one argument u can also remove the parenthesis in args

Return statement is also not required

The screenshot shows an IDE window titled "OCJA 1.8 Java SE 8 Programmer - I (170 - 808) by Durga Sir On 08-04-2018". The code editor displays the following Java code:

```

20
21
22
23 eg3:
24 public int squareIt(int x)
25 {
26     return x*x;
27 }
28
29 (int x)->{return x*x;}
30 (int x)->return x*x;
31 (x)->return x*x;
32 x->return x*x;
33 x->x*x;

```

The IDE interface includes a toolbar at the top with icons for file operations, a status bar at the bottom showing "Ln 33, col 8, 33, 00, PC, ANSI", and a video player control bar at the very bottom.

If only one line then curly brackets are optional

Multi lines curly brackets are needed

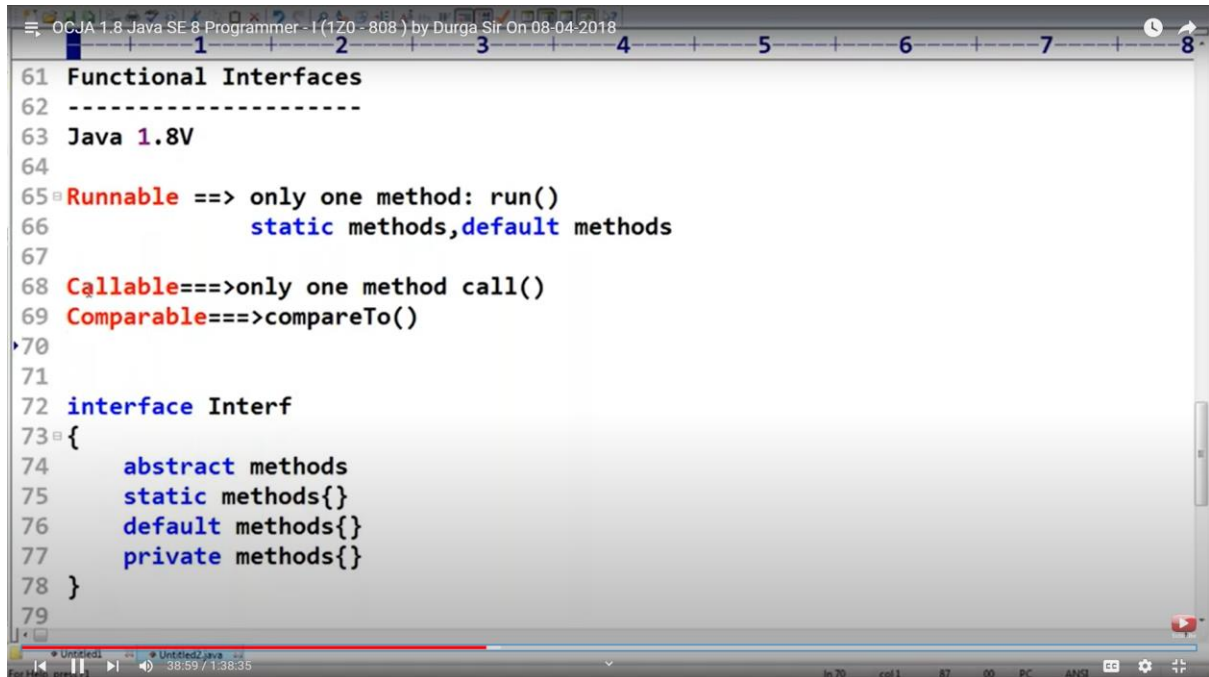
```
OCJA 1.8 Java SE 8 Programmer - I (170 - 808 ) by Durga Sir On 08-04-2018
1 2 3 4 5 6 7 8
34
35 Conclusions:
36 -----
37 1. Any number of arguments
38 2. Not required to specify the type
39 3. parameters separated with ,
40 4. zero no of parameters
41 () ->
42 {
43     System.out.println("Hello");
44     System.out.println("Hello");
45     System.out.println("Hello");
46     System.out.println("Hello");
47 }
48 5. x->x*x
56
57 (a,b,c)->a+b+c;
```

Lambda can be called using Functional Interface

If an interface contains only 1 abstract method its called Functional Interface.

It may contain any no of Static methods, default and Private Methods does not matter. If it has only 1 abstract method its called Functional Interface

Runnable Interface has only 1 Abstract Method.

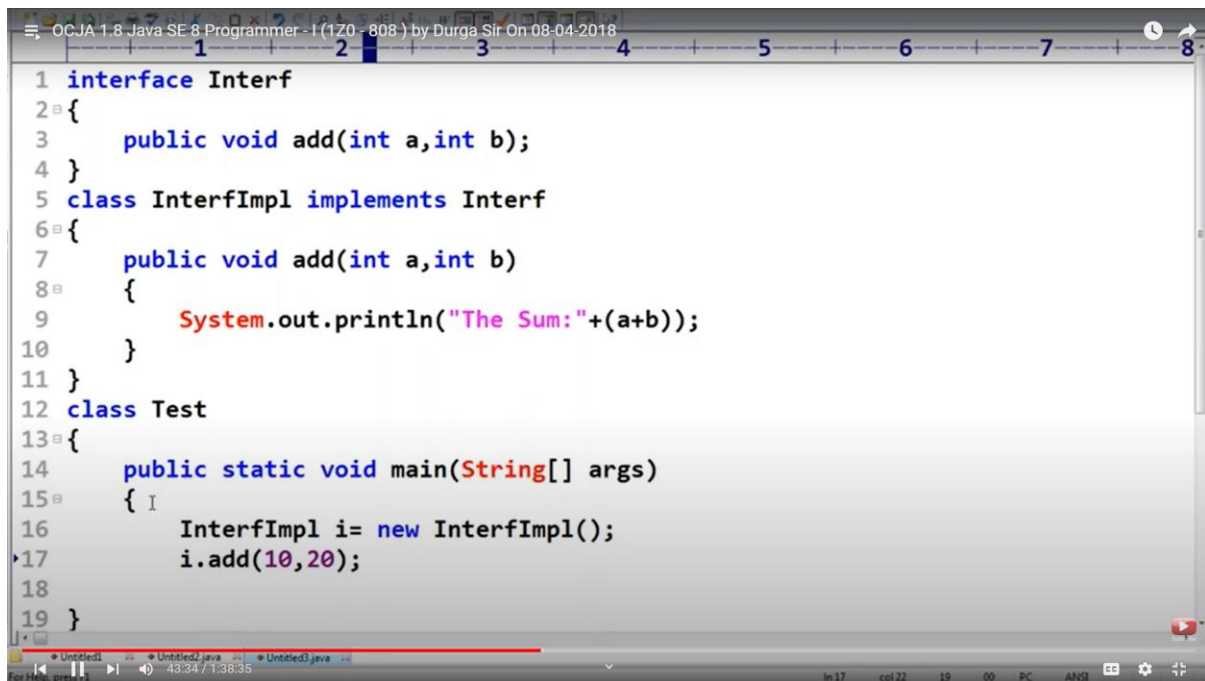


The screenshot shows a video player window with a presentation slide titled "Functional Interfaces". The slide content is as follows:

```
61 Functional Interfaces
62 -----
63 Java 1.8V
64
65 Runnable ==> only one method: run()
66           static methods, default methods
67
68 Callable ==> only one method call()
69 Comparable ==> compareTo()
70
71
72 interface Interf
73 {
74     abstract methods
75     static methods{}
76     default methods{}
77     private methods{}
78 }
79
```

The video player interface includes a progress bar at the bottom, a timestamp of 38:59 / 1:00:35, and a status bar showing "Ln 70 col 1 87 00 PC ANS".

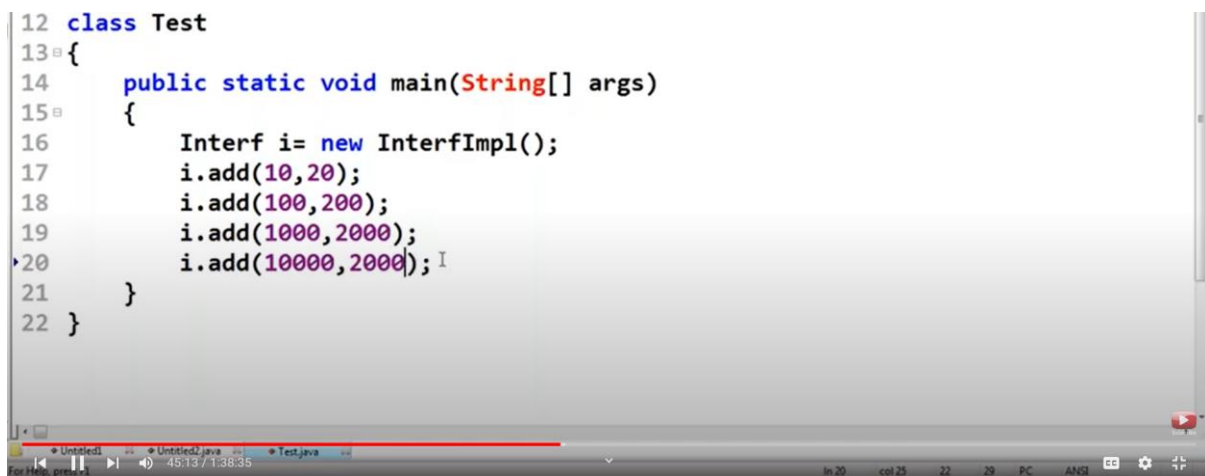
There is an interface and class implements it and we create an object and call it



```
1 interface Interf
2 {
3     public void add(int a,int b);
4 }
5 class InterfImpl implements Interf
6 {
7     public void add(int a,int b)
8     {
9         System.out.println("The Sum:"+(a+b));
10    }
11 }
12 class Test
13 {
14     public static void main(String[] args)
15     {
16         InterfImpl i= new InterfImpl();
17         i.add(10,20);
18     }
19 }
```

The screenshot shows a code editor window titled "OCJA 1.8 Java SE 8 Programmer - I (1Z0-808) by Durga Sir On 08-04-2018". The code defines an interface 'Interf' with a method 'add'. A class 'InterfImpl' implements this interface. A 'Test' class contains a 'main' method that creates an instance of 'InterfImpl' and calls the 'add' method with arguments 10 and 20. The IDE interface includes a toolbar at the top with icons for file operations and a status bar at the bottom showing 'Ln 17 col 22 19 00 PC ANS'.

Once an object is created u can call the method many times



```
12 class Test
13 {
14     public static void main(String[] args)
15     {
16         Interf i= new InterfImpl();
17         i.add(10,20);
18         i.add(100,200);
19         i.add(1000,2000);
20         i.add(10000,2000);
21     }
22 }
```

This screenshot shows the 'Test' class from the previous example, but with multiple calls to the 'add' method. The 'main' method now calls 'i.add' with four different pairs of arguments: (10, 20), (100, 200), (1000, 2000), and (10000, 2000). The IDE interface is similar to the first screenshot, with a status bar at the bottom showing 'Ln 20 col 25 22 29 PC ANS'.

(a,b) are args to add method as my reference is Interf. This is a lambda expression, implementing functional interface that contains 1 abstract method

```
1 interface Interf
2 {
3     public void add(int a,int b);
4 }
5 class Test
6 {
7     public static void main(String[] args)
8     {
9         Interf i=(a,b)->System.out.println("The Sum:"+(a+b));
10        i.add(10,20);
11        i.add(100,200);
12        i.add(1000,2000);
13        i.add(10000,20000);
14    }
15 }
```

```
1 interface Interf
2 {
3     public int squareIt(int x);
4 }
5 class Test
6 {
7     public static void main(String[] args)
8     {
9         Interf i=x->x*x;
10        System.out.println(i.squareIt(10));
11        System.out.println(i.squareIt(20));
12        System.out.println(i.squareIt(30));
13    }
14 }
15 }
```

```
1 class MyRunnable implements Runnable
2 {
3     public void run()
4     {
5         for(int i =0; i<10;i++)
6         {
7             System.out.println("Child Thread");
8         }
9     }
10 }
11 class Test
12 {
13     public static void main(String[] args)
14     {
15         MyRunnable r = new MyRunnable();
16         Thread t = new Thread(r);
17         t.start();
18         for(int i =0; i<10; i++)
19         {
20             System.out.println("Main Thread");
21         }
22     }
23 }
```

```
1 class Test
2 {
3     public static void main(String[] args)
4     {
5         Runnable r = ()->
6         {
7             for(int i=0;i<10;i++)
8                 System.out.println("Child Thread By Lambda");
9         };
10         Thread t = new Thread(r);
11         t.start();
12         for(int i =0; i<10; i++)
13         {
14             System.out.println("Main Thread");
15         }
16     }
17 }
```



```
1 class Test
2 {
3     public static void main(String[] args)
4     {
5         Runnable r = ()->{ for(int i=0;i<10;i++) System.out.println("Child Threa
6         Thread t = new Thread(r);
7         t.start();
8         for(int i =0; i<10; i++)
9         {
10             System.out.println("Main Thread");
11         }
12     }
13 }
```

OCJA 1.8 Java SE 8 Programmer - I (170 - 808) by Durga Sir On 08-04-2018

For Help, press F1

ln 5 col 1 13 09 PC ANSI

```
40 Boolean valued function
41
42 Predicate(I)==> Functional interface
43
44 boolean test(T t)
45
46 interface Predicate<T>
47 {
48     public boolean test(T t);
49 }
50
51
52
53
54
55
56
57
58
```

OCJA 1.8 Java SE 8 Programmer - I (170 - 808) by Durga Sir On 08-04-2018

For Help, press F1

ln 48 col 30 70 20 PC ANSI

```
OCJA 1.8 Java SE 8 Programmer - I (1Z0-808) by Durga Sir On 08-04-2018
1 interface Predicate<T>
2 {
3     public boolean test(T t);
4 }
5 class PredicateImpl implements Predicate
6 {
7     public boolean test(Integer i)
8     {
9         if(i>10)
10        {
11            return true;
12        }
13        else
14        {
15            return false;
16        }
17    }
18 }
```

```
OCJA 1.8 Java SE 8 Programmer - I (1Z0-808) by Durga Sir On 08-04-2018
1 import java.util.function.*;
2 class Test
3 {
4     public static void main(String[] args)
5     {
6         Predicate<Integer> p= i->i>10;
7         System.out.println(p.test(100)); //true
8         System.out.println(p.test(5)); //false
9     }
10 }
```

```
OCJA 1.8 Java SE 8 Programmer - I (170 - 808) by Durga Sir On 08-04-2018
1 import java.util.function.*;
2 class Test
3 {
4     public static void main(String[] args)
5     {
6         Predicate<String> p=s->s.length()>3;
7         System.out.println(p.test("Shubham"));
8         System.out.println(p.test("abc"));
9     }
10 }
11 }
```

```
OCJA 1.8 Java SE 8 Programmer - I (170 - 808) by Durga Sir On 08-04-2018
1 import java.util.function.*;
2 import java.util.*;
3 class Test
4 {
5     public static void main(String[] args)
6     {
7         Predicate<Collection> p=c->c.isEmpty();
8         ArrayList l = new ArrayList();
9         System.out.println(p.test(l));
10 }
11 }
```

```
1 import java.util.function.*;
2 import java.util.*;
3 class Test
4 {
5     public static void main(String[] args)
6     {
7         Predicate<Collection> p=c->c.isEmpty();
8         ArrayList l = new ArrayList();
9         l.add("Durga");
10        System.out.println(p.test(l));
11    }
12 }
```

For Help, press F1

```
OCJA 1.8 Java SE 8 Programmer - I (170 - 808) by Durga Sir On 08-04-2018
73
74
75 Predicate Joining:
76 -----
77
78 and()
79 or()
80 negate()
81
82
83
84
85
86
87
```

For Help, press F1

```
OCJA 1.8 Java SE 8 Programmer - I (120 - 808 ) by Durga Sir On 08-04-2018
1 2 3 4 5 6 7 8
4 public static void main(String[] args)
5 {
6     int[] x = {0,5,10,15,20,25,30};
7     Predicate<Integer> p1=i->i>10;
8     Predicate<Integer> p2=i->i%2==0;
9     System.out.println("Numbers Greater than 10:");
10    m1(p1,x);
11    System.out.println("Even Numbers are:");
12    m1(p2,x);
13 }
14 public static void m1(Predicate<Integer> p,int[] x)
15 {
16     for(int x1: x)
17     {
18         if(p.test(x1))
19         {
20             System.out.println(x1);
21         }
22     }
23 }
```

To print nos not greater than 10

```
1 2 3 4 5 6 7 8
4 public static void main(String[] args)
5 {
6     int[] x = {0,5,10,15,20,25,30};
7     Predicate<Integer> p1=i->i>10;
8     Predicate<Integer> p2=i->i%2==0;
9     System.out.println("Numbers Greater than 10:");
10    m1(p1,x);
11    System.out.println("Even Numbers are:");
12    m1(p2,x);
13    System.out.println("The Numbers not Greater than 10:");
14    m1(p1.negate(),x);
15 }
16 public static void m1(Predicate<Integer> p,int[] x)
17 {
18     for(int x1: x)
19     {
20         if(p.test(x1))
21         {
22             System.out.println(x1);
23         }
24     }
25 }
```

```
10      m1(p1,x);
11      System.out.println("Even Numbers are:");
12      m1(p2,x);
13      System.out.println("The Numbers not Greater than 10:");
14      m1(p1.negate(),x);
15
16      System.out.println("Numbers Greater than 10 and even:");
17      m1(p1.and(p2),x);
18
19  }
20  public static void m1(Predicate<Integer> p,int[] x)
21  {
22      for(int x1: x)
23      {
24          if(p.test(x1))
25          {
26              System.out.println(x1);
27          }
28      }
29  }
```

For Help, press F1

```
10      m1(p1,x);
11      System.out.println("Even Numbers are:");
12      m1(p2,x);
13      System.out.println("The Numbers not Greater than 10:");
14      m1(p1.negate(),x);
15
16      System.out.println("Numbers Greater than 10 and even:");
17      m1(p1.and(p2),x);
18
19      System.out.println("Numbers Greater than 10 or even:");
20      m1(p1.or(p2),x);
21
22  }
23  public static void m1(Predicate<Integer> p,int[] x)
24  {
25      for(int x1: x)
26      {
27          if(p.test(x1))
28          {
29              System.out.println(x1);
30          }
31      }
32  }
```

For Help, press F1