Lambda Expressions (From Java 1.8)

→ Java doesn't support functional programming.
→ To support functional programming in java we use Lambda Expressions.
→ Code Optimization.

→ Lambda Function is an Anonymous Function (nameless function)
→ Anonymous function —
- nameless function
- No return type
- No access modifier

ex –

```
public void main(){
  Sop("Hello");
}
```

→

```
() -> {
  Sop("Hello");
}
```

→ we only have one statement so, we can remove { }

```
() -> Sop("Hello");
```

ex –

```
public void m1(int a, int b)
{  Sop(a+b);
}
```

→

```
(int a, int b) -> Sop(a+b);
```

No need to specify type

↓

```
(a, b) -> Sop(a+b);
```

→ Create lambda Expression for functions returning values.

```
public void m1(int a)
{  return (a*a);
}
```

→

```
(int a) -> {return (a*a); }
```

↓ Single parameter, so optional

```
(a) -> a*a;
```

↓

```
a -> a*a;
```

Check for valid lamba expression

1. `n -> return n*n;` ✗
2. `n -> {return n*n;};` ✓ curly braces required when using return keyword.
3. `n -> {return n*n;};` ✗ semicolon is missing
4. `n -> {n*n;};` ✗ Curly braces not required, when not using return keyword.
5. `n -> n*n;` ✓

How to Invoke Lambda expressions.

Functional Interface

Interface – contains only Abstract methods.
Abstract Methods – contains declaration but not body of Method.

ex-
```
interface Player {
    final int id = 10;
    int move();
}
```

Java 1.7 ↓
Functional Interface :- contains only Single Abstract Method. (SAM)

→ from Java 1.7, Interface can store :-
- Abstract methods
- default methods
- Static methods

}

→ Functional Interface can have only one Abstract method but can have any number of default and static methods.

Default Functional Interfaces:-

1) Runnable → run()
2) Callable → call()
3) Comparable → compareTo()
4) ActionListner → actionPerformed()

→ Lambda expressions can be invoked only through functional Interface

Explicitly specify function Interface :-

Use annotation : @FunctionalInterface

ex-
```
@FunctionalInterface
interface I {
    public void m1();
    default void m2() {}
}
```
✓

Without Using Lambda Expression ⇒

```java
package lambdaexpression;

@FunctionalInterface
interface Cab{
    public void bookCab();
}

class Ola implements Cab {

    public void bookCab() {
        System.out.println("Ola booked");
    }
}

public class Test {
    public static void main(String[] args) {
        Cab cab = new Ola();
        cab.bookCab();
    }
}

```

→ Print Using Lambda Expression https://www.linkedin.com/in/kapilyadav22

```java
package lambdaexpression;


@FunctionalInterface
interface Cab2{
    public void bookCab();

    default public void city() {
        System.out.println("delhi");
    }
}

public class lambda2 {
    public static void main(String[] args) {
        Cab2 cab = () -> System.out.println("Ola booked");
        cab.bookCab();
        cab.city();
    }
}

```

→ Using Lambda Expression

Without Using Lambda Expression

```java
package lambdaexpression;


@FunctionalInterface
interface Cab3{

    public void bookCab(String Source, String Destination);
    default public void city() {
        System.out.println("delhi");
    }
}

class Ola implements Cab3{
    public void bookCab(String Source, String Destination) {
        System.out.println("Cab is booked from"+Source+" to "+Destination);
    }
}

public class lambda3 {
    public static void main(String[] args) {
        Cab3 cab = new Ola();
        cab.bookCab("Delhi", "Rajasthan");
        cab.city();
    }
}
```

Without Using Lambda Expression

-> Passing Parameters in Lambda Expression

```java
package lambdaexpression;


@FunctionalInterface
interface Cab3{

    public void bookCab(String Source, String Destination);
    default public void city() {
        System.out.println("delhi");
    }
}

public class lambda3 {
    public static void main(String[] args) {
        Cab3 cab =(Source,Destination)
          -> System.out.println("cab is booked from "+ Source +" to "+Destination);
        cab.bookCab("Delhi", "Rajasthan");
        cab.city();
    }
}
```

taking Arguements in Lambda expression

Without Using Lambda Expression

```java
package lambdaexpression;


@FunctionalInterface
interface Cab4{
    public String bookCab(String Source, String Destination);
}

class Ola implements Cab4{
    public String bookCab(String Source, String Destination) {
        System.out.println("cab is booked from "+ Source +" to "+Destination);
        return ("Price : Rs 3000");
    }
}
public class lambda4 {
    public static void main(String[] args) {
        Cab4 cab = new Ola();
        String Price = cab.bookCab("Delhi", "Rajasthan");
        System.out.println(Price);

    }
}


```

→ Return value Using Lambda Expression.

```java
package lambdaexpression;

@FunctionalInterface
interface Cab4{
    public String bookCab(String Source, String Destination);
}

public class lambda4 {
    public static void main(String[] args) {
        Cab4 cab =(Source,Destination) ->
        { System.out.println("cab is booked from "+ Source +" to "+Destination);
            return ("Price : Rs 3000");
        };

        String Price = cab.bookCab("Delhi", "Rajasthan");
        System.out.println(Price);

    }
}
```

Pre-defined functional Interface

Java 1.8

→ Even though, we don't have Functional Interface, we can use Lambda expressions using Pre-defined functional Interface.

Some of the Pre-defined Functional Interface (java.util.Function)

1. Predicate<T> — test()
2. Function<T,R> — apply()
3. Consumer<T> — accept()
4. Supplier<R> — get()

1. Predicate Interface :- contains test() abstract Method.

```
interface Predicate<T>{
    public boolean test(T t);
}
```

→ test() contains only 1 arguement (Can be of any type) and always return boolean value.

```java
package predicate;

import java.util.function.Predicate;

//Predicate Interface can be use when we have some conditional statements
public class PredicateExample {

    public static void main(String[] args) {

        Predicate<Integer> p = i->(i>20);
        System.out.println(p.test(30)); //true
        System.out.println(p.test(20)); //false

    }
}
```

→ Predicate

```java
package predicate;

import java.util.function.Predicate;

//Predicate Interface can be use when we have some conditional statements
public class PredicateExample2 {

    public static void main(String[] args) {

        //check the length of string, whether is greater than 4 or not
        Predicate<String> p = s->(s.length()>4);
        System.out.println(p.test("kapil")); //true
        System.out.println(p.test("abc")); //false
    }
}
```

```java
package predicate;

import java.util.function.Predicate;

//Predicate Interface can be use when we have some conditional statements
public class PredicateExample3 {

    public static void main(String[] args) {

        //print array elements whose size is greater than 4

        String names[] = {"Kapil", "Rahul", "Vineet", "Ravi", "Rohit"};

        Predicate<String> p = s->(s.length()>4);
        for(String name: names) {
                if(p.test(name)) {
                    System.out.println(name);
                }
        }
    }
}
```

```java
package predicate;

import java.util.function.Predicate;


class Employee{
    String employeeName;
    Float salary;
    int experience;

    public Employee(String employeeName, Float salary, int experience) {
        super();
        this.employeeName = employeeName;
        this.salary = salary;
        this.experience = experience;
    }
}

public class PredicateExample4 {

    public static void main(String[] args) {

        //create lambda expression for objects
        Predicate<Employee> p = e->(e.salary>50000 && e.experience>3);

        Employee emp = new Employee("Kapil", 75000f, 0);
        Employee emp1 = new Employee("Rahul", 15000f, 5);
        Employee emp2 = new Employee("Ravi", 30000f, 3);
        Employee emp3 = new Employee("Kartik", 145000f, 4);
        Employee emp4 = new Employee("VIrat", 14542000f, 15);

        System.out.println(p.test(emp));  //false
        System.out.println(p.test(emp1)); //false
        System.out.println(p.test(emp2)); //false
        System.out.println(p.test(emp3)); //true
        System.out.println(p.test(emp4)); //true
    }
}

```

```java
package predicate;

import java.util.ArrayList;
import java.util.function.Predicate;


class Employee{
    String employeeName;
    Float salary;
    int experience;

    public Employee(String employeeName, Float salary, int experience) {
        super();
        this.employeeName = employeeName;
        this.salary = salary;
        this.experience = experience;
    }
}

public class PredicateExample4 {

    public static void main(String[] args) {

        //create lambda expression for objects
        Predicate<Employee> p = e->(e.salary>50000 && e.experience>3);

        ArrayList<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Kapil", 75000f, 0));
        employees.add(new Employee("Rahul", 15000f, 5));
        employees.add(new Employee("Kartik", 145000f, 4));
        employees.add(new Employee("Virat", 14542000f, 15));
        employees.add(new Employee("Rohit", 75000656f, 16));
        employees.add(new Employee("KL Rahul", 75045600f, 4));

        for(Employee employee: employees) {
            if(p.test(employee)) {
                System.out.println(employee.employeeName+"    "+ employee.salary);
            }

        }

    }
}

```

```java
package predicate;

import java.util.function.Predicate;

//Joining Predicates - and, or, negate
    //p1 && p2
    // p1 || p2
    //p1.negate().test()


//p1 -> check, if number is evem
//p2 -> checks, number>50

public class PredicateExample5 {

    public static void main(String[] args)      {
        int arr[] = {10,20,33,24,23,5,3,53,35,85,54,60,70,78};

        Predicate<Integer> p1 = n->n%2==0;
        Predicate<Integer> p2 = n->n>50;

        //and predicates
        for(int num : arr) {
            //if(p1.test(num) && p2.test(num)) {
            if(p1.and(p2).test(num)) {
                System.out.println(num);
            }
        }
        System.out.println("\n Or Predicate");

        //or Predicates
        for(int num : arr) {
            //if(p1.test(num) || p2.test(num)) {
            if(p1.or(p2).test(num)) {
                System.out.println(num);
            }

        }

        System.out.println("\n negate Predicate");
        for(int num : arr) {
            if(p1.negate().test(num)) {
                System.out.println(num);
            }

        }
    }
}

```

Function Interface

→ function Interface contains apply() method.
→ Predicate Interface returns bool type, but for returning values we can use function Interface.
→ It will return a single value of any type.

ex –

```
interface Function<T,R>
{
    public R apply(T);
}
```

(Return type)

→ Predicate ⟶ ParameterType → returns Boolean
→ test()
→ and, or, negate

Function → Parameter Type, Return Type → returns some type
→ apply()
→ andThen() → f1, then f2.
→ compose() → f2, then f1

```
f1.andThen(f2).apply(—);
f1.compose(f2).apply(—);
```

```java
package functionInterface;

import java.util.function.Function;

public class functionExample1 {

    public static void main(String[] args) {

        Function<Integer, Integer> f = n->n*n;

        System.out.println(f.apply(10));
        System.out.println(f.apply(20));
        System.out.println(f.apply(30));
        System.out.println(f.apply(40));
        System.out.println(f.apply(50));
        System.out.println(f.apply(60));
    }

}

OUTPUT:

    100
    400
    900
    1600
    2500
    3600
```

```java
package functionInterface;

import java.util.function.Function;

public class functionExample2 {

    public static void main(String[] args) {
        Function<String, Integer> f = s->s.length();

        System.out.println(f.apply("Kapil"));
        System.out.println(f.apply("Rahul"));
        System.out.println(f.apply("Vineet"));
        System.out.println(f.apply("Abhishek"));
        System.out.println(f.apply("aditya"));
        System.out.println(f.apply("Alok"));

    }
}
```

```java
package functionInterface;

import java.util.ArrayList;
import java.util.function.Function;
import java.util.function.Predicate;

class Employee {
    String employeeName;
    Float salary;

    public Employee(String employeeName, Float salary) {
        super();
        this.employeeName = employeeName;
        this.salary = salary;
    }
}

public class functionExample3 {

    public static void main(String[] args) {

        ArrayList<Employee> employees = new ArrayList<Employee>();

        employees.add(new Employee("Kapil", 75000f));
        employees.add(new Employee("Rahul", 15000f));
        employees.add(new Employee("Kartik", 145000f));
        employees.add(new Employee("Virat", 14542000f));
        employees.add(new Employee("Rohit", 75000656f));

        Function<Employee, Float> fn = e -> {
            Float salary = e.salary;

            if (salary > 10000 && salary <= 1000000) {
                return (salary * 0.1f);
            } else if (salary > 100000 && salary < 200000) {
                return (salary * 0.2f);
            } else
                return (salary * 0.3f);
        };
        //use Predicate to check, whether the bonus is greater than 50000 or not
        Predicate<Float> p = bn -> bn > 50000;

        for (Employee emp : employees) {
            float bonus = fn.apply(emp);
          //print only if bonus > 50000;
            if (p.test(bonus))
                System.out.println("Employee Salary is : " + emp.salary + " and bonus is : " + bonus);
        }
    }

}

```

```java
package functionInterface;

import java.util.function.Function;

//Function Chaining
    //andThen()
    //compose
public class functionChaining {

    public static void main(String[] args) {
        //<arguementtype,returntype>
        Function<Integer, Integer> f1 = n->n*2;
        Function<Integer, Integer> f2 = n->n*n;

        //using andThen, first f1 will execute and the result of f1 will pass to f2
        System.out.println(f1.andThen(f2).apply(10)); //400

        //using compose, first f2 will execute and then f1 will execute
        System.out.println(f2.andThen(f1).apply(10)); //200
    }

}
```

Consumer Interface

→ Consumer Interface contains accept() method, which takes one input (single parameter) but doesn't return anything.
→ It uses/consumes the input, we pass to it.

Supplier Interface

→ It contains get() method.
→ It will not take any parameter, but it will supply some value/object. (returns object/elements)

```java
package consumer;

import java.util.function.Consumer;

public class consumerExample1 {

    public static void main(String[] args) {

        Consumer<String> c = s-> System.out.println(s);

        c.accept("Hello Everyone");
    }

}

```

```java
package consumer;

import java.util.ArrayList;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Predicate;

class Employee {
    String employeeName;
    Float salary;

    public Employee(String employeeName, Float salary) {
        super();
        this.employeeName = employeeName;
        this.salary = salary;
    }
}

public class consumerExample2 {

    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<Employee>();

        employees.add(new Employee("Kapil", 75000f));
        employees.add(new Employee("Rahul", 15000f));
        employees.add(new Employee("Kartik", 145000f));
        employees.add(new Employee("Virat", 14542000f));
        employees.add(new Employee("Rohit", 75000656f));

        Function<Employee, Float> fn = e -> {
            Float salary = e.salary;

            if (salary > 10000 && salary <= 1000000) {
                return (salary * 0.1f);
            } else if (salary > 100000 && salary < 200000) {
                return (salary * 0.2f);
            } else
                return (salary * 0.3f);
        };

        Predicate<Float> p = bn -> bn > 50000;

        Consumer<Employee> c = e -> System.out
                .println("Employee Name is : " + e.employeeName +
                        "and his Salary is : " + e.salary);

        for (Employee emp : employees) {
            float bonus = fn.apply(emp);
            if (p.test(bonus)) {
                c.accept(emp);
                System.out.println("Bonus is" + bonus);
            }
        }
    }
}
```

```java
package consumer;

import java.util.function.Consumer;

//consumerchaining
public class consumerChaining {

    public static void main(String[] args) {

        Consumer<String> c1 = s-> System.out.println(s+" is White");
        Consumer<String> c2 = s-> System.out.println(s+" have 4 legs");
        Consumer<String> c3 = s-> System.out.println(s+" eats grass");

//      c1.accept("Cow");
//      c2.accept("Cow");
//      c3.accept("Cow");

        c1.andThen(c2).andThen(c3).accept("Cow");
        //or

        Consumer<String> c4 = c1.andThen(c2).andThen(c3);
        c4.accept("Cow");
    }

}

```

SUPPLIER

```java
package supplier;

import java.util.Date;
import java.util.function.Supplier;

public class supplierExample1 {

    public static void main(String[] args) {

        Supplier<Date>  s = () -> new Date();
        System.out.println(s.get());
    }
}
```