Writing Lambda for a Functional interface :

```java
package com.ravi.lambda;

@FunctionalInterface
interface Predictable
{
    boolean verify(int num);
}

public class LambdaExample
{
    public static void main(String[] args)
    {
        Predictable p1 =   num -> num % 2==0;

        int no = Integer.parseInt(IO.readln("Enter a number :"));

        boolean isEven = p1.verify(no);

        IO.println("Is "+no+" is even number :"+isEven);

    }

}
```

Working with predefined functional interfaces :

* In order to work with **day to day programming** and to write **concise code, Java software people has provided the following predefined functional interfaces.**

* These **predefined functional interfaces** are available in **java.util.function** sub package.

1) Predicate<T>
2) Consumer<T>
3) Function<T,R>
4) Supplier<T>
5) BiPredicate<T,U>
6) BiConsumer<T,U>
7) BiFunction<T,U,R>
8) UnaryOperator<T>
9) BinaryOperator<T, T>

Predicate<T> functional interface :

* It is a predefined functional interface available in java.util.function sub package.

```java
@FunctionalInterface
public interface Predicate<T>
{
  boolean test(T x);
}
```

* It is used to **verify one argument boolean expression.**

* It accepts one abstract method test() which is accepting T as a parameter and return type is boolean.

//Program

```java
package com.ravi.predicate_demo;

import java.util.function.Predicate;

//Based on the age, verify whether a person is eligible for voting or not?

public class PredicateDemo1 {

    public static void main(String[] args)
    {
        Predicate<Integer> p1 =  age -> age>=18;

        int myAge = Integer.parseInt(IO.readln("Enter your age :"));
        IO.println("Are u eligible for vote :"+p1.test(myAge));
    }

}
```

PredicateDemo2.java

```java
package com.ravi.predicate_demo;

import java.util.function.Predicate;

//Verify whether my name is Ravi Or not

public class PredicateDemo2
{
    public static void main(String[] args)
    {
        Predicate<String> p2 = name -> name.equalsIgnoreCase("Ravi");

        String myName = IO.readln("Enter your Name :");
        IO.println("Are you Ravi :"+p2.test(myName));
    }
}
```

PredicateDemo3.java

```java
package com.ravi.predicate_demo;

import java.util.function.Predicate;

class Employee
{
    private int id;
    private String name;
    private double salary;


    public Employee(int id, String name, double salary)
    {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
    }



    public int getId()
    {
        return id;
    }



    public String getName()
    {
        return name;
    }



    public double getSalary()
    {
        return salary;
    }



    @Override
    public String toString()
    {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary +
"]";
    }

}


public class PredicateDemo3
{
    public static void main(String[] args)
    {
         //Verify whether employee salary is > 50 or not [Working with custom
object]

        Predicate<Employee> p3 = emp -> emp.getSalary() > 50000;
        boolean isvalid = p3.test(new Employee(1, "Scott", 40000));
        IO.print("Your salary is greater than 50K :"+isvalid);

    }

}
```

PredicateDemo4.java

```java
package com.ravi.predicate_demo;

import java.util.function.Predicate;

class Student
{
    private int id;
    private String name;
    private double marks;

    public Student(int id, String name, double marks)
    {
        super();
        this.id = id;
        this.name = name;
        this.marks = marks;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getMarks() {
        return marks;
    }

    public void setMarks(double marks) {
        this.marks = marks;
    }
}


public class PredicateDemo4 {

    public static void main(String[] args)
    {
        //Based on the marks verify whether the student is pass OR fail
        //Marks > 65 -> pass otherwise Fail

        Predicate<Student> p4 = std -> std.getMarks() > 65;

        Student []students = new Student[5];
        students[0] = new Student(1, "Scott", 60);
        students[1] = new Student(2, "Smith", 72);
        students[2] = new Student(3, "John", 75);
        students[3] = new Student(4, "Alen", 68);
        students[4] = new Student(5, "Raj", 62);

        for(Student student : students)
        {
            boolean isPass = p4.test(student);

            if(isPass)
            {
                IO.println(student.getName()+" is Pass ");
            }
            else
            {
                IO.println(student.getName()+" is Fail ");
            }

        }

    }

}
```

```java
default Predicate<T> and(Predicate<? super T> other)
```

* Predicate interface has provided a default method called and()
* This and() method is used to combine two predicates
* This method return type is Predicate<T> functional interface

```java
package com.ravi.predicate_demo;

import java.util.function.Predicate;

public class PredicateDemo5 {

    public static void main(String[] args)
    {
        int num = Integer.parseInt(IO.readln("Enter a number :"));

        Predicate<Integer> isPositive = num1 -> num1>0;
        Predicate<Integer> isEven = num2 -> num2%2==0;

        //Combine both the Predicate using and()
        Predicate<Integer> and = isPositive.and(isEven);
        IO.println("Is the number is positive & even :"+and.test(num));

    }

}
```

Consumer<T> functional interface :

* It is a predefined functional interface available in java.util.function sub package.

```java
@FunctionalInterface
public interface Consumer<T>
{
  void accept(T x);
}
```

* It is mainly used to consume OR accept the value.

* It accepts accept() abstract method and does not return anything

* It takes parameter of type T

```java
package com.ravi.consumer;

import java.util.function.Consumer;

public class ConsumerDemo {

    public static void main(String[] args)
    {
        Consumer<String> c1 = str -> IO.println("String type is :"+str);
        c1.accept("Java");

        Consumer<Integer> c2 = num -> IO.println("Integer Type is :"+num);
        c2.accept(12);

        Consumer<Customer> c3 = cust -> IO.println("Customer type is :"+cust);
        c3.accept(new Customer());
    }

}

class Customer
{
    @Override
    public String toString()
    {
        return "Customer Object";
    }
}
```