```
interfaces of jdk1.8V
++++++++++++++++++++
public interface java.util.function.Predicate<T>
{

  public abstract boolean test(T);

  public default java.util.function.Predicate<T> and(java.util.function.Predicate<?
super T>);
  public default java.util.function.Predicate<T> negate();
  public default java.util.function.Predicate<T> or(java.util.function.Predicate<?
super T>);

  public static <T> java.util.function.Predicate<T> isEqual(java.lang.Object);
  public static <T> java.util.function.Predicate<T>
not(java.util.function.Predicate<? super T>);
}


Note:
It is an functional interface present in java.util.function package.
It is an interface which contains only one abstract method called test(T).
Since the interface Predicate is functional interface,we can invoke the method
using "LambdaExpression".

Write a Predicate to check wheter the given integer is greater than 10 or not?

Traditional Approach(OOPS)
++++++++++++++++++++++++++
public boolean test(Integer i)
{
      if(i>10)
      {
            return true;
      }
      else
      {
            return false;
      }
}

Lambda Expression
+++++++++++++++++
i -> i>10;


eg#1.
import java.util.function.*;
public class Test{
      public static void main(String[] args){
            //Binded to test(Integer) :: boolean
            Predicate<Integer> p = i->i>10;

            System.out.println("Result is :: "+p.test(10));
            System.out.println("Result is :: "+p.test(100));
            System.out.println("Result is :: "+p.test(-5));
            System.out.println("Result is :: "+p.test("sachin"));//CE
      }
}
```

```
Output
Result is :: false
Result is :: true
Result is :: false
```

eg#2.
Write a predicate to check the length of the given String is greater than 3 or not.
Input :: MI, RCB,CSK,LSG,RR

eg#1.

```
import java.util.function.*;
public class Test{
      public static void main(String[] args){
            //Binded to test(Integer) :: boolean
            Predicate<String> p = name-> name.length() >=3;

            System.out.println("Result is :: "+p.test("MI"));
            System.out.println("Result is :: "+p.test("CSK"));
            System.out.println("Result is :: "+p.test("RCB"));
            System.out.println("Result is :: "+p.test("RR"));
      }
}
Output
Result is :: false
Result is :: true
Result is :: true
Result is :: false
```

eg#3.
Write a predicate to check whether the given Array number is less than 18 or not?

```
 int[] arr = {10,20,30,40,50,60};
```

eg#4.
Write a predicate to check whether the given Array elements are even in number or not?

```
 int[] arr = {10,20,30,40,50,60};
```

Approach

```
++++++++
import java.util.function.*;
public class Test{
      public static void main(String[] args){

            int[] arr = {0,5,10,15,20,25,30};

            //Binded to test(Integer) :: boolean
            Predicate<Integer> p1= i -> i < 18;
            Predicate<Integer> p2= i -> i%2 == 0;

            System.out.print("Numbers which are less than 18::  ");
            performOperation(p1,arr);

            System.out.println();

            System.out.print("Numbers which are even ::  ");
            performOperation(p2,arr);
```

```
            System.out.println();
      }

      public static void performOperation(Predicate<Integer>p,int[] arr)
      {
            for (int data:arr)
            {
                  if (p.test(data))
                  {
                        System.out.print(data+"\t");
                  }
            }
            System.out.println();
      }
}
```

Predicate Joining
++++++++++++++++
It is possible to join predicates into single predicate by using the following methods

a. and()
b. or()
c. negate()

These are exactly same as logical operators like AND,OR,NOT

eg#1.

```
import java.util.function.*;
public class Test{
      public static void main(String[] args){

            int[] arr = {0,5,10,15,20,25,30};

            //Binded to test(Integer) :: boolean
            Predicate<Integer> p1= i -> i < 18;
            Predicate<Integer> p2= i -> i%2 == 0;

            System.out.print("Numbers which are less than 18::  ");
            performOperation(p1,arr);

            System.out.println();

            System.out.print("Numbers which are even ::  ");
            performOperation(p2,arr);

            System.out.println();

            System.out.print("Numbers which are not less than 18::  ");
            performOperation(p1.negate(),arr);

            System.out.println();

            System.out.print("Numbers which are less than 18 and even numbers::
");
            performOperation(p1.and(p2),arr);

            System.out.println();

            System.out.print("Numbers which are less than 18 or even numbers::  ");
            performOperation(p1.or(p2),arr);
```

```
            System.out.println();

      }

      public static void performOperation(Predicate<Integer>p,int[] arr)
      {
            for (int data:arr)
            {
                  if (p.test(data))
                  {
                        System.out.print(data+"\t");
                  }
            }
            System.out.println();
      }
}
```

```
Output
Numbers which are less than 18::  0     5       10      15

Numbers which are even ::  0    10      20      30

Numbers which are not less than 18::  20        25      30

Numbers which are less than 18 and even numbers::  0    10

Numbers which are less than 18 or even numbers::  0     5       10      15      20
30
```

Write a predicate to check whether the age of student is less than 30 or not?

eg#1.

```
import java.util.function.*;

class Student
{
      private String name;
      private Integer age;

      Student(String name,Integer age)
      {
            this.name = name;
            this.age  = age;
      }

      public String getName(){
            return name;
      }
      public Integer getAge(){
            return age;
      }
}
public class Test{
      public static void main(String[] args){

            Student[] std = new Student[3];
            Student std1 = new Student("sachin",25);
            Student std2 = new Student("dravid",31);
```

```
            Student std3 = new Student("kohli",28);
            std[0] = std1;
            std[1] = std2;
            std[2] = std3;

            Predicate<Student> p = student -> student.getAge()<30;
            performOperation(p,std);

      }

      public static void performOperation(Predicate<Student>p,Student[] students)
      {
            int count = 0;
            for (Student student: students )
            {
                  if(p.test(student))
                        count++;
            }
            System.out.println("No of students whose age is less < 30 is ::
"+count);
      }
}
```

Output
No of students whose age is less < 30 is :: 2

Function
+++++++++
 => They are exactly same as Predicate, except that functions can return any type of Result.
 => It is present inside a package called "java.util.function.Function"
 => It contains only one method called " R apply(T)".
 => Since Function is a FunctionalInterface, we can refer it through "Lambda Expression".

```
public interface java.util.function.Function<T, R> {

  public abstract R apply(T);

  public default <V> java.util.function.Function<V, R>
compose(java.util.function.Function<? super V, ? extends T>);
  public default <V> java.util.function.Function<T, V>
andThen(java.util.function.Function<? super R, ? extends V>);

  public static <T> java.util.function.Function<T, T> identity();
}
```

Write a function to display the length of the given String?

```
public int apply(String data)
{
      int count=data.length();
      return count;
}
```

Lambda Expression
+++++++++++++++++
```
data-> data.length();
```

eg#1.

```
import java.util.function.*;
public class Test{
      public static void main(String[] args){

            Function<String,Integer> f = s-> s.length();
            System.out.println("Length of the String is :: "+f.apply("dhoni"));
            System.out.println("Length of the String is :: "+f.apply("sachin"));
      }
}
```

Output

```
Length of the String is :: 5
Length of the String is :: 6
```

Write a Function to get the result as half of the given supplied input?

```
import java.util.function.*;
public class Test{
      public static void main(String[] args){

            Function<Integer,Double> f = i->i/2.0;
            System.out.println(f.apply(10));
            System.out.println(f.apply(5));
      }
}
```

Output

```
5.0
2.5
```

Chaining

++++++++

eg#1.

```
import java.util.function.*;

public class Test{
      public static void main(String[] args){

            Function<String,Integer> f1  = s-> s.length();
            Function<Integer,Integer> f2 = i->i*2;

            System.out.println(f1.andThen(f2).apply("sachin"));
            System.out.println(f1.andThen(f2).apply("RCB"));
      }
}
```

Output

```
12
6
```

eg#2.

```
import java.util.function.*;
public class Test{
      public static void main(String[] args){

            Function<Integer,Double>  f1  = i->i/2.0;
```

```
            Function<Integer,Integer> f2  = i->i*3;
            System.out.println(f1.compose(f2).apply(5));
      }
}
```

Output

```
7.5
```