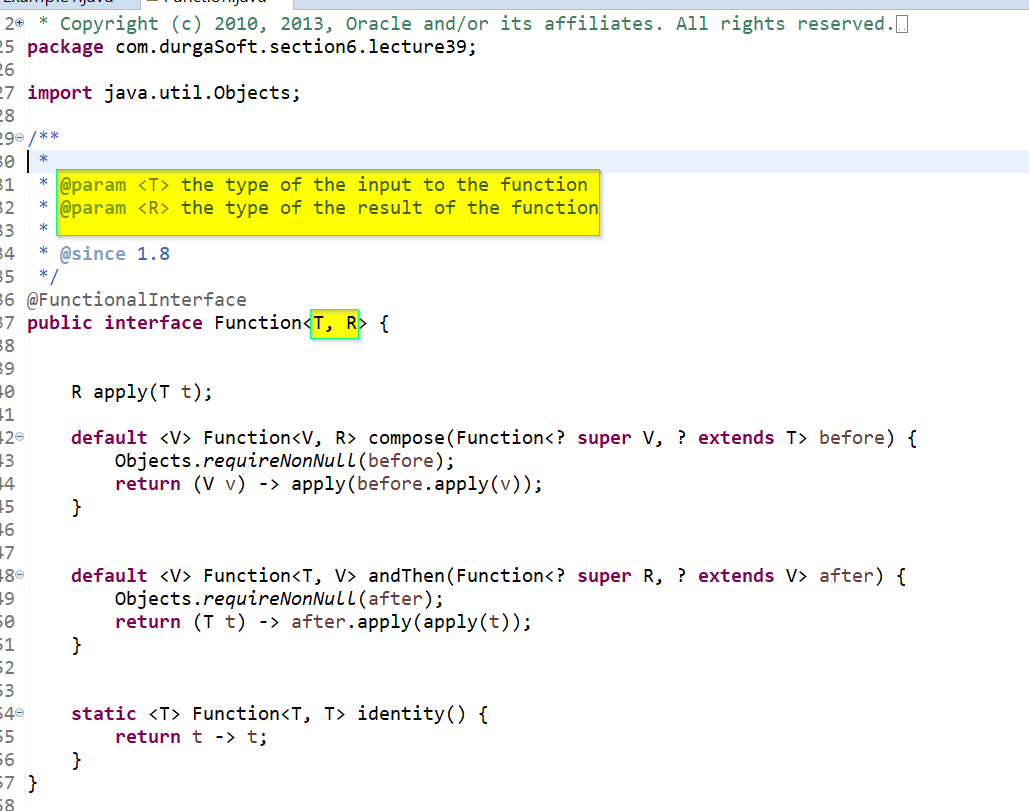
Lecture 39: Predefined Functional Interface- Function:

We can even tell that the predicate is a special case of function where the return type is always boolean

Function is similar to predicate except it can return any value:



Write a code to find the length of the String:

**package** com.durgaSoft.section6.lecture39;

//To calculate the lenght of a given String

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Function<String,Integer> lengthOfString = (s) -> s.length();

System.***out***.println(lengthOfString.apply("Prashanth"));

System.***out***.println(lengthOfString.apply("We"));

}

}

|  |  |  |
| --- | --- | --- |
| S.I | Predicate | Function |
|  |  |  |
| 1 | We should go for predicate when we need to perform conditional checks | We should go for Function when we need to perform certain operations and to return some result. |
|  |  |  |
| 2 | Predicate can take one type parameter which represents input argument type.  Predicate<T> | Function can take 2 type parameters. First one represents input argument type and second one represent return type.  Function<T,R> |
|  |  |  |
| 3 | Predicate interface define one abstract method called test() | Function interface defines one abstract method called apply() |
|  |  |  |
| 4 | public boolean test(T t) | public R apply(T t) |
|  |  |  |
| 5 | Predicate can return only boolean value. | Function can return any type of value |

Lecture 40:

Program to remove space in a string:

**package** com.durgaSoft.section7.lecture40;

**import** java.util.function.Function;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Function<String, String> removeSpace = (string) -> string.replaceAll(" ","");

String message ="Durga Software Solutions Hyderabad";

System.***out***.println(removeSpace.apply(message));

}

}

To count the number of space in a string:

**package** com.durgaSoft.section7.lecture40;

**import** java.util.function.Function;

/\*

\* Here I am using indexOf method which will check for a particular character in a

\* String.

\*/

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

String message = "Durga Software Solutions Hyderabad";

Function<String, Integer> spaceCounter = string -> {

**int** result=0; **int** count=0;

**for**(;result!=-1;count++) {

result =string.indexOf(" ", result+1);

}

//Count should be subtracted by 1 as its even counting when the value is -1

**return** count-1;

};

System.***out***.println(spaceCounter.apply(message));

}

}

Another brilliant method.

**public** **class** Example3 {

**public** **static** **void** main(String[] args) {

String message = "Durga Software Solutions Hyderabad";

Function<String, Integer> spaceCounter = string ->

message.length() - message.replaceAll(" ","").length();

System.***out***.println(spaceCounter.apply(message));

}

}

Lecture 43:

To perform salary increment:

**package** com.durgaSoft.section7.lecture47.example1;

**import** java.util.ArrayList;

**import** java.util.function.Function;

**import** java.util.function.Predicate;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

ArrayList<Employee> empList = **new** ArrayList<Employee>();

*pupulateEmployee*(empList);

Predicate<Employee> salaryCheck = emp -> emp.salary < 3500;

Function<Employee, Employee> empSalIncrement = employee -> {

employee.salary = employee.salary + 477;

**return** employee;

};

**for** (Employee employee : empList) {

**if** (salaryCheck.test(employee)) {

empSalIncrement.apply(employee);

}

}

System.***out***.println(empList);

}

**private** **static** **void** pupulateEmployee(ArrayList<Employee> employee) {

// **TODO** Auto-generated method stub

employee.add(**new** Employee("Sunny", 1000));

employee.add(**new** Employee("Bunny", 2000));

employee.add(**new** Employee("Chinny", 3000));

employee.add(**new** Employee("Pinny", 4000));

employee.add(**new** Employee("Vinny", 5000));

}

}

Lecture 44:

Function Chaining:

* Consider the two functions f1 and f2
* andThen: f1.andThen(f2) -> f1 will be applied followed by f2.
* f1.compose(f2) -> f2 will be applied followed by f1.

How to perform Function Chaining.

**package** com.durgaSoft.section7.lecture44;

**import** java.util.function.Function;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Function<String, String> toUpperCase = s -> s.toUpperCase();

Function<String, String> subString = s -> s.substring(0,9);

System.***out***.println(toUpperCase.apply("AiswaryaAbhi"));

System.***out***.println(subString.apply("AiswaryaAbhi"));

System.***out***.println(toUpperCase.andThen(subString).apply("AiswaryaAbhi"));

System.***out***.println(toUpperCase.compose(subString).apply("AiswaryaAbhi"));

}

}

Lecture 45:

Difference between andThen and compose:

**package** com.durgaSoft.section7.lecture45;

**import** java.util.function.Function;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Function<Integer, Integer> f1 = i -> i+i;

Function<Integer, Integer> f2 = i -> i\*i\*i;

System.***out***.println(f1.andThen(f2).apply(2)); //64

System.***out***.println(f1.compose(f2).apply(2)); //16

}

}

Lecture 47:

**package** com.durgaSoft.section7.lecture47;

**import** java.util.function.Function;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Function<String, String> f1 = Function.*identity*();

System.***out***.println(f1.apply("Prashanth")); //Prashanth

}

}