Using predicate for filtering collection

As in previous post we saw that Predicate is used to filter the collection. It has a test() method that is used to evaluate a predicate.

In this post we will see how to use Predicate interface for filtering collection.

Let us assume that we have a class Person.

**package** com.example.javase8.filtercollections;

**public** **class** Person {

**private** String name;

**private** **int** age;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "[name=" + name + ", age=" + age + "]";

}

}

Now let us add person in a list with their name and age.

List<Person> list=**new** ArrayList<>();

list.add(**new** Person("Ned",10));

list.add(**new** Person("Ryan",20));

list.add(**new** Person("James",30));

list.add(**new** Person("Roy",40));

list.add(**new** Person("Joshua",50));

Now let us define test method. We want those people whose age is greater than 20.

Normally we would write that code like this.

**for** (Person p : list) {

**if** (p.getAge() > 20) {

System.***out***.print(p + " ");

}

}

With Predicate interface we would write like this.

Predicate<Person> obj = **new** Predicate<Person>() {

@Override

**public** **boolean** test(Person p) {

**return** (p.getAge() > 20);

}

};

We just wrote the test method. And what happened? You expected a lambda operator or lambda expression. Right? Yes. We will write that too. But first let us understand what this thing is doing.

Now we will call Predicate’s test method to display those objects in list that pass the criteria “Age must be greater than 20”.

Predicate<Person> **obj** = **new** Predicate<Person>() {

@Override

**public** **boolean** test(Person p) {

**return** (p.getAge() > 20);

}

};

**for** (Person person : list) {

**if** (**obj**.test(person)) {

System.***out***.print(person);

}

}

Yes so the above code will do the work. But the thing is we want to use lambda expression and it just so happens that Predicate interface is a Functional Interface. Now let us add some lambdas.

Predicate<Person> predOlder = (p) -> p.getAge() > 20;

Now we will use forEach(..) method and filter the collection as per the Predicate.

list.forEach(p -> {

**if** (predOlder.test(p)) {

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

}

});

Output

[name=James, age=30]

[name=Roy, age=40]

[name=Joshua, age=50]

Well, this is nice. We just defined a predicate and used in forEach(..) loop as an action.

Let us take one more step and introduce something new for this post. We used forEach(..) method to filter the collection based on test method of Predicate of type person. Don’t forget the Predicate is of type generic.

Let me just throw the code here.

list.stream()

.filter(predOlder)

.forEach(p->System.***out***.println(p));

stream() is the method that returns a stream for the specified collection. filter(..) method accepts the predicate and returns the stream that accepts this predicate. (You now understand why I was using word **filter** frequently). And then we just use forEach(..) method to print the filtered person object in list.

This methods and syntax may seem bit wired. But don’t worry we will cover stream afterwards. It is good topic to explore. But that comes later.