Day 2: Functional Interface in Java 8 (Predicate, Supplier, Consumer, Function)

Some of the new functional interfaces introduced in java 8 to perform functional style of programming. these interfaces belongs to java.util.function package.

1. **Predicate<T>**
2. **Consumer<T>**
3. **Supplier<T>**
4. **Function<T,R>**

1. **java.util.function.Predicate<T>:**

   This interface contains only one abstract method called:

   ```
   public boolean test(T t);
   ```

   This method test() checks whether supplied obj satisfying a condition or not.

   Example:

   ```
   import java.util.function.Predicate;
   public class Main {

           public static void main(String[] args) {

               Predicate<Integer> p = i -> i > 0;

               System.out.println(p.test(10));//true
               System.out.println(p.test(-10));//false
   ```

```
        }
}
```

In java 8 Collection interface defines a method called:

`public boolean removeIf(Predicate filter)`

Based on the condition of Predicate, this method will remove/filter the elements from the Collection classes

Example: Removing the Students from the List whose marks is less than 700

```
import java.util.List;
import java.util.ArrayList;
public class Main{

  public static void main(String[] args)   {

    List<Student> students=new ArrayList<>();

    students.add(new Student(10, "name1", 650));
    students.add(new Student(12, "name2", 750));
    students.add(new Student(13, "name3", 550));
    students.add(new Student(14, "name4", 820));
    students.add(new Student(15, "name5", 720));
    students.add(new Student(16, "name6", 620));


    System.out.println(students);

    students.removeIf( student -> student.getMarks() < 700 );

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

2. **java.util.function.Consumer<T>:**

   It represents a function which takes in one argument and produces a result. However these kind of functions don’t return any value.

```
public void accept(T t);
```

Example:

```
import java.util.function.Consumer;
public class Main {

        public static void main(String[] args) {

            Consumer<Student> c = s -> {

                System.out.println("Roll is "+s.getRoll());
                System.out.println("Name is "+s.getName());
                System.out.println("Marks is "+s.getMarks());
            };

            c.accept(new Student(10,"Amit",850));
        }
}
```

Note :- from java 8 each collection classes contains a method called

```
default void forEach(Consumer c);
```

This forEach method defined as **default** method inside the **java.lang.Iterable** interface.

Example: iterating elements of a List using forEach method.

```
import java.util.ArrayList;
import java.util.List;
public class Main{
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Football");
        list.add("Cricket");
        list.add("Chess");
        lList.add("Hocky");


        list.forEach(s -> System.out.println(s));

        //or using Method referensce
        //list.forEach(System.out::println);

    }
}
```

3.java.util.function.Supplier<T>:

It represents a function which does not take in any argument but produces a value of type T.

method:

```
public T get();
```

Example:

```
import java.util.function.Supplier;
public class Main {

        public static void main(String[] args) {

            Supplier<String> s = () -> "This is from Lambda Expression";

            System.out.println(s.get());

            Supplier<Student> s2 = () -> new Student(10,"Ram",850);

            System.out.println(s2.get().getName());

        }
}
```

4.java.util.function.Function<T,R>

This interface defines an abstract method which will takes T type of object as parameter and returns R type of object.

```
public R apply(T t);
```

Example:

```
import java.util.function.Function;

public class Main {

        public static void main(String[] args) {

            Function<Integer,String> f = i -> "This is a numner "+i;

            System.out.println(f.apply(10));

            Function<String,Integer> f2 = s -> Integer.parseInt(s);
```

```
            System.out.println(f2.apply("200")+500);

            Function<String,Integer> f3 = Integer::parseInt;
            System.out.println(f3.apply("400")+200);
        }
}
```

Example2: Getting a Student object and returning greeting message with Student Name.

```
import java.util.function.Function;
public class Main {

        public static void main(String[] args) {


            Function<Student,String> f = s -> "Welcome "+s.getName().toUpperCase();

            String msg= f.apply(new Student(10,"Amit",850));

            System.out.println(msg);

        }
}
```