Predefined Functional Interfaces

Predicate<T>

We can use Predicate<T> to implement some conditional checks. However, from it's method signature : boolean test(T t) it is clear that it takes an input parameter and returns a Boolean result. When you have this type of requirement to write a method, use it confidently. Let's observe the method signature as below:

```
public interface Predicate<T> {

   boolean test(T t);

}
```

=> T denotes the input parameter type.

Function<T, R>

Function<T, R> is used to perform some operation & returns some result. Unlike Predicate<T> which returns only boolean, Function<T, R> can return any type of value. Therefore, we can also say that Predicate is a special type of Function which returns only Boolean values.

```
interface Function<T,R> {

   R apply(T t);

}
```

=> T is method input parameter & R is return type

Consumer<T>

Consumer<T> is used when we have to provide some input parameter, perform certain operation, but don't need to return anything. Moreover, we can use Consumer to consume object and perform certain operation.

```
interface Consumer<T> {

  void accept(T t);

}
```

=> T denotes the input parameter type.

Supplier<R>

Supplier<R> doesn't take any input and it always returns some object. However, we use it when we need to get some value based on some operation like supply Random numbers, supply Random OTPs, supply Random Passwords etc.

For example, below code denotes it :

```
interface Supplier<R>{

   R get();

}
```

=> R is a return type

BiPredicate<T, U>

BiPredicate<T, U> is same as Predicate<T> except that it has two input parameters.

For example, below code denotes it:

```
interface BiPredicate<T, U> {
   boolean test(T t, U u)
}
```

=> T & U are input parameter types

BiFunction<T, U, R>

BiFunction<T, U, R> is same as Function<T, R> except that it has two input parameters. For example, below code denotes it:

```
interface BiFunction<T, U, R> {
   R apply(T t, U u);
}
```

=> T & U are method input parameters & R is return type

BiConsumer<T, U>

BiConsumer<T> is same as Consumer<T> except that it has two input parameters. For example, below code denotes it:

```
interface BiConsumer<T, U> {
  void accept(T t, U u);
}
```

=> T & U are method input parameters