Functional Interface and lambda

- in java 8 minimal code feature and functional programming has been introduced
- upgrades in java 8 → lambda expression, Stream , Date and time API

Lambda Expression

- it is a anonymous function which do not have any name, return type or access modifier.
- Lambda expression is used to implement functional interface(single abstract method).
- syntax :: (param in funcn interface method) → { logic };
- code to use functional interface using lambda

```
@FunctionalInterface
public interface MathOperation{

public int operate(int a , int b);
}
```

following class is ignored when using lambda

```
public class Addition implements MathOperation{

@Override
public int operate(int a, int b){
retun a+b;
}

}
```

```
public class demo{

public static void main (String[] args){

step 1 ->

(int a, int b)->{
retun a+b;
}

step 2->

MathOperation sum = (int a, int b) -> {
retun a+b;
}
```

```
16
17 step 3 -> more optimised
18 MathOperation sum = ( a ,  b) -> a+b; // functional programming
19 MathOperation subs = ( a ,  b) -> a-b;
20 int result = sum.operate(1,2);
21 System.out.println(result);
22
23 }
24 }
```

Predicate Interface [functional interface]

- Boolean valued function which has method named as test(T t);
- Predicate functional interface holds the condition
- we are storing condition in variable, this is functional programming
-
```
Predicate<Integer> isEven = x- > x%2 ==0;
System.out.println(isEven.test(2));   //true

Predicate<Integer> doStartWithA = x- > x.startsWith("A");
Sout(doStartWithA.test("Suyash")); //False

Predicate<Integer> doEndWithA = x- > x.endssWith("A");
Sout(doStartWithA.test("Suyash")); //False

Predicate<String> and = doStartWithA.and(doEndWithA);
sout(and.test("Suyash"));
```

Function interface[Functional interface]

- it has abstract method 'apply(T t)'
- Function has a method which holds a function eg multiply by2, divvide by 2.
- Function stores a logic for you mostly calculation

```
Function<Integer,Integer> doubeit =  x-> x*2;
Function<Integer,Integer> trippleIt =  x-> x*3;
Syso(doubleit.apply(2));
Syso(doubleit.andThen(trippleIt).apply(50));
```

Consumer interface [Functional interface]

- it has abstract method 'accept(T t)' of void return type.
- It just consumes do not produce anything
- Mostly used to do syso using lambda

```
Consumer<integer> print = x -> System.out.println(x);
print.accept("Hello");
```

```
List<Integer> list = Arrays.asList(1,2,3,5);
Consumer<List<Integer>> printList = x -> {
for(int a : x){
system.out.println(a);
}
printlist.accept(list);

}
```

Supplier Interface[Functional interface]

- It has a get() abstract method which consumes nothing in the parenthesis but produces result.
- ```
  Supplier<String> prints = () -> "Hello world";
  Sout(prints.get());
  ```

Combined Example

```
Predicate<Integer> predicate = x -> x%2 ==0;
Function<Integer,Integer> funcation = x -> x*x;
Consumer<Integer> consumer = x -> System.out.rintln(x);
Supplier<Integer> supplier =  () -> 100;

if(predicate.test(supplier.get())){
consumer.accept(function.apply(supplier.get()));
}
```

BiPredicate, BiConsumer, BiFunction

```
BiPredicate<Integer,Integer> isSumEven = (x,y) -> (x+y) % 2 == 0;
Sysyo(isSumEven.test(12,26));

BiConsumer<Integer,String> bi = (a,b) -> {
syso(a);
syso(b);
};
bi.accept(2, "michael");

BiFunction<String,String , Integer> bfun = (a,b) -> (a+b).length();
sout(bfun.apply("abc","uvw"));
```

⭐ Conclusion Table →

Functional Interface	Abstract method name	No of input parameters in abstr	return type	Example
```

		act method		
Predicate	test(T t);	1	boolean	`Predicate<Integer> p = (a) -> a % 2 == 0;`
Function	apply(T t);	1	user defined	`Function<Integer, Integer> f = (x) -> x * 2;`
Consumer	accept(T t)	1	void	`Consumer<String> c = y -> System.out.println(y);`
Supplier	get();	0	user defined	`Supplier<String> sup = () -> "Supplier";`
BiPredicate	test(T t,U u)	2	boolean	`BiPredicate<Integer, Integer> bp = (x,y) -> (x+y)%2 ==0;`
BiConsumer	accept(T t,U u)	2	void	`BiConsumer<String, String> bic = (x,y) -> { System.out.println("String a " +x); System.out.println("String b " +y); }; bic.accept("hello", "world");`
BiFunction	apply(T t,U u)	2	user defined	`BiFunction<Integer, Integer, Integer> Bifun = (a,b) -> (a+b)/2; int a= Bifun.apply(2, 10); System.out.println(a);`

Document by ***Suyash*** 😇