5.2. BiPredicate Functional Interfaces

- **BiPredicate** Functional Interface
  - Takes Two Input Parameters
  - Returns Boolean after processing.
- **BiPredicate** Functional Interface has the following methods.

abstract boolean **test**(T, U);

BiPredicate<T, U> **and**(BiPredicate<? super T, ? super U>);

BiPredicate<T, U> **negate**();

BiPredicate<T, U> **or**(BiPredicate<? super T, ? super U>);

Demo8: Files Required:

1. Demo8.java	

Demo8.java

```
package com.jlcindia.demos;

import java.util.function.BiPredicate;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo8 {
public static void main(String[] args) {

BiPredicate<Integer,Integer> predicate1 = (num1,num2) -> num1>num2;

boolean mybool = predicate1.test(10,20);
System.out.println(mybool);

BiPredicate<Integer,Integer> predicate2 = (num1,num2) -> num1<num2;
mybool = predicate2.test(10,20);
System.out.println(mybool);

}
}
```

5.3. Function Functional Interfaces

- **Function** Functional Interface
  - Takes One Input Parameter
  - Returns Output after processing.
- **Function** Functional Interface has the following methods.

```
abstract R apply(T); //SAM
static <T> Function<T, T> identity();
<V> Function<V, R> compose(Function<? super V, ? extends T>);
<V> Function<T, V> andThen(Function<? super R, ? extends V>);
```

Demo9: Files Required:

1. Demo9.java	

Demo9.java

```
package com.jlcindia.demos;

import java.util.function.Function;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo9 {
public static void main(String[] args) {

Function<String, String> fun1 = (input) -> input.toUpperCase();

String output1 = fun1.apply("Srinivas Dande");
System.out.println(output1);

Function<String, Integer> fun2 = (input) -> Integer.parseInt(input);

Integer output2 = fun2.apply("99");
System.out.println(output2);

Function<Integer, String> fun3 = (input) ->  String.valueOf(input);

String output3 = fun3.apply(99);
System.out.println(output3);
```

```
Function<String, String> fun4 = input -> input;

String output4= fun4.apply("Hello Guys");
System.out.println(output4);

Function<String, String> fun5 = Function.identity();

String output5= fun5.apply("Hello Guys");
System.out.println(output5);
}
}
```

Demo10: Files Required:

1. Demo10.java	

Demo10.java

```
package com.jlcindia.demos;

import java.util.function.Function;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Demo10 {
public static void main(String[] args) {

Function<Integer, Integer> fun1 = (num) -> {
System.out.println("Multiply by 2");
return num * 2;
};

Function<Integer, Integer> fun2 = (num) -> {
System.out.println("Multiply by 3");
return num * 3;
};

System.out.println(fun1.apply(5));
System.out.println(fun2.apply(5));

int result2= fun1.andThen(fun2).apply(10);
//fun1.apply(10) => 10 * 2 => 20
//fun2.apply(20) => 3 * 20 => 60
System.out.println(result2); //60
```

```
int result1= fun1.compose(fun2).apply(10);
//fun2.apply(10) => 10 * 3 => 30
//fun1.apply(30) =>30 * 2 =>  60
System.out.println(result1); //60
}
}
```

Demo11: Files Required:

1. Demo11.java	

Demo11.java

```
package com.jlcindia.demo1;

import java.util.function.Function;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo11 {
public static void main(String[] args) {

Function<String, Integer> fun1 = (input) -> {
System.out.println("Converting String to Integer");
return Integer.parseInt(input);
};

Function<Integer, Integer> fun2 = (num) -> {
System.out.println("add 10 to the number");
return num + 10;
};

int result2= fun1.andThen(fun2).apply("10");
System.out.println(result2); //20

int result1= fun2.compose(fun1).apply("10");
System.out.println(result1); //20
}
}
```

5.4. UnaryOperator Functional Interfaces

- **UnaryOperator** Functional Interface is sub type of Function which allows you specify only One Type for both parameter and Return Value.
- **UnaryOperator** Functional Interface
  - Takes One Input Parameter
  - Returns Output after processing.
- **UnaryOperator** Functional Interface has the following methods.

```
abstract R apply(T); //SAM
public static <T> UnaryOperator<T> identity();
<V> Function<V, R> compose(Function<? super V, ? extends T>);
<V> Function<T, V> andThen(Function<? super R, ? extends V>);
```

Demo12.java

```
package com.jlcindia.demos;

import java.util.function.Function;
import java.util.function.UnaryOperator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo12 {
public static void main(String[] args) {

Function<String,String> fun1 = (input) -> input.toUpperCase();
System.out.println(fun1.apply("Hello"));

UnaryOperator<String> unary1 = (input) -> input.toUpperCase();
System.out.println(unary1.apply("Hello"));

Function<Integer, Integer> fun2 = (num) -> num * 2;
System.out.println(fun2.apply(50));

UnaryOperator<Integer> unary2 = (num) -> num * 2;
System.out.println(unary2.apply(50));
}
}
```

5.5. BiFunction Functional Interfaces

- **BiFunction** Functional Interface
  - Takes Two Input Parameters
  - Returns Output after processing.
- **BiFunction** Functional Interface has the following methods.

```
abstract R apply(T, U); //SAM
<V> BiFunction<T, U, V> andThen(Function<? super R, ? extends V>);
```

Demo13.java

```
package com.jlcindia.demos;

import java.util.function.BiFunction;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo13 {
public static void main(String[] args) {

BiFunction<String,String,String> fun1 = (input1,input2) -> input1 + input2;
String output = fun1.apply("Hello"," Guys");
System.out.println(output);

BiFunction<Integer, Integer,Integer> fun2 = (num1,num2) -> num1 * num2;
System.out.println(fun2.apply(5,25));

BiFunction<Integer, Integer,String> fun3 = (num1,num2) -> {
int result= num1 * num2;
String str = "Result is "+result;
return str;
};
System.out.println(fun3.apply(5,25));
}
}
```

5.6. BinaryOperator Functional Interfaces

- **BinaryOperator** Functional Interface is sub type of BiFunction which allows you specify only One Type for both parameter and Return Value.
- **BinaryOperator** Functional Interface
  - Takes Two Input Parameters
  - Returns Output after processing.
- **BinaryOperator** Functional Interface has the following methods.

```
static <T> BinaryOperator<T> minBy(Comparator<? super T>);
static <T> BinaryOperator<T> maxBy(Comparator<? super T>);
```

Demo14.java

```
package com.jlcindia.demos;

import java.util.function.BiFunction;
import java.util.function.BinaryOperator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo14 {

public static void main(String[] args) {

BiFunction<String,String,String> fun1 = (input1,input2) -> input1 + input2;
String output = fun1.apply("Hello"," Guys");
System.out.println(output);

BinaryOperator<String> binary1 = (input1,input2) -> input1 + input2;
String output1 = binary1.apply("Hello"," Guys");
System.out.println(output1);

BiFunction<Integer, Integer,Integer> fun2 = (num1,num2) -> num1 * num2;
System.out.println(fun2.apply(5,25));

BinaryOperator<Integer> binary2 = (num1,num2) -> num1 * num2;
System.out.println(binary2.apply(5,25));
}
}
```

5.7. Consumer and BiConsumer Interfaces

- **Consumer** Functional Interface
  - Takes One Input Parameter
  - No Return Value.
- **Consumer** Functional Interface has the following methods.

  abstract void accept(T); //SAM

  Consumer<T> andThen(Consumer<? super T>);

- **BiConsumer** Functional Interface
  - Takes Two Input Parameters
  - No Return Value.
- **BiConsumer** Functional Interface has the following methods.

  abstract void accept(T, U);

  BiConsumer<T, U> andThen(BiConsumer<? super T, ? super U>);

Demo15.java

```
package com.jlcindia.demos;

import java.util.function.BiConsumer;
import java.util.function.Consumer;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo15 {
public static void main(String[] args) {

Consumer<String> consumer1 = (input) -> System.out.println(input.toUpperCase());

consumer1.accept("Hello");
consumer1.accept("Srinivas");
consumer1.accept("Hai");

BiConsumer<String,String> consumer2 = (input1,input2) ->
System.out.println(input1+input2);
consumer2.accept("Hello"," Guys");
}
}
```

5.8. Supplier Functional Interfaces

- **Supplier** Functional Interface
  - Takes No Input Parameters
  - Returns Any Type .
- **Supplier** Functional Interface has the following methods.

  abstract T get();

Demo16.java

```
package com.jlcindia.demos;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.util.function.Supplier;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Demo16 {
public static void main(String[] args) {

Supplier<String> supplier1 = () -> "Hello Guys, How are you?";

String str = supplier1.get();
System.out.println(str);

Supplier<Integer> supplier2 = () ->  LocalDate.now().getDayOfMonth();
System.out.println(supplier2.get());

Supplier<DayOfWeek> supplier3 = () ->  LocalDate.now().getDayOfWeek();
DayOfWeek dow = supplier3.get();

System.out.println(dow);
System.out.println(dow.getValue());

}
}
```