BridgeLabz

Employability Delivered

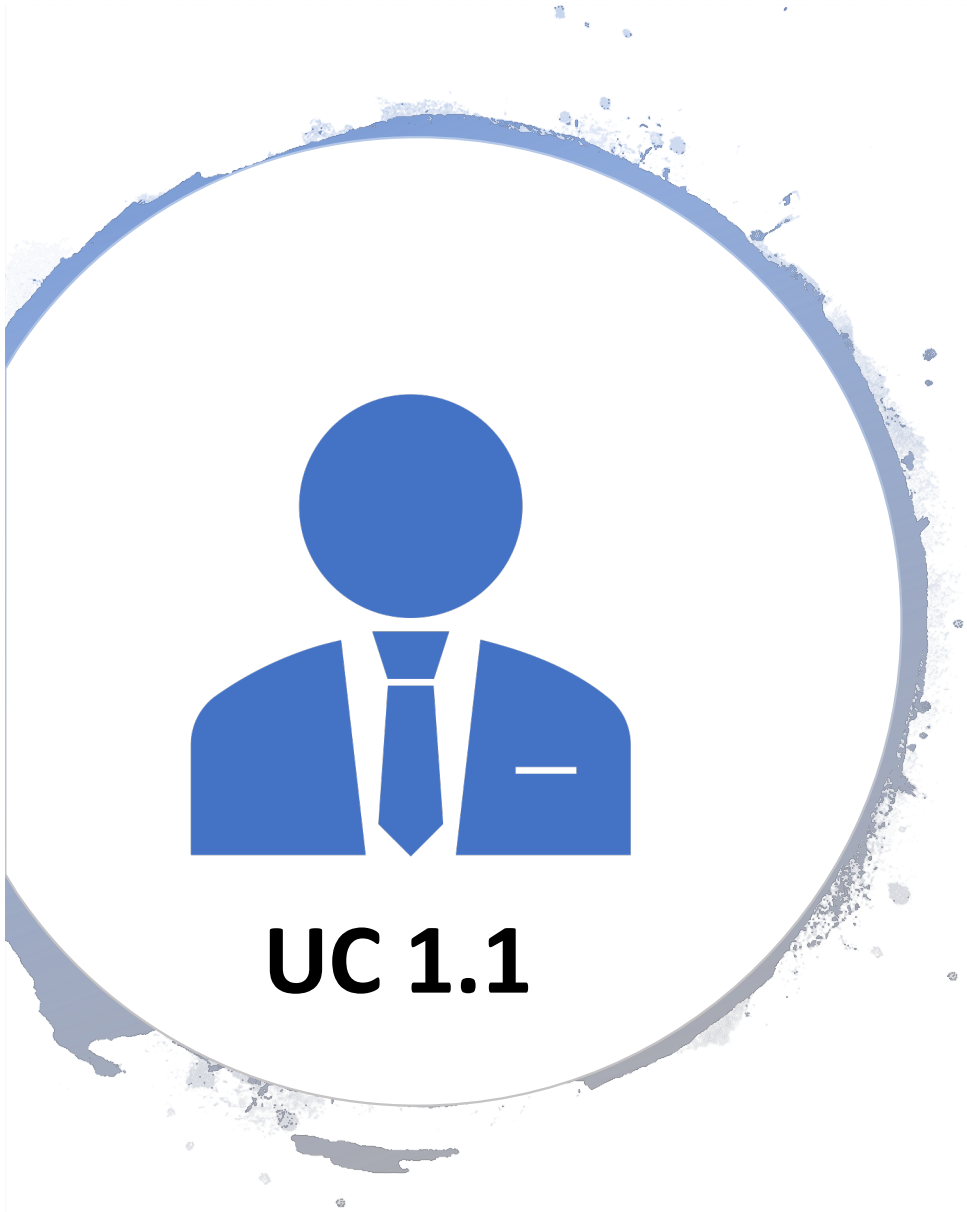
Java8 Core Concepts

Section 1: Lambda Expression

Functional Programming

BridgeLabz
Employability Delivered

- Programming Languages is defined by certain syntactical and structural norms. These norms are called **Programming Paradigms**. The different paradigms are:
  - Imperative, Object-oriented, Functional, Logic, and so forth.
- **Functional programming** is one of many such paradigms. It emphasizes on declarative aspects of programming where business logic is composed of pure functions, an idea that somewhat contrasts the essence of object-oriented methodology which is essentially Objects and its behavior.
- Java imbibed the power of Functional Programming using **Functional Interface** and **Lambda Expression**



Develop Math Operation App to perform Math Functions – Addition, Subtraction and Division

- Use Lambda Expression to perform Math Operation

Functional Interface

BridgeLabz
Employability Delivered

- Functional Interfaces are also called **Single Abstract Method interfaces** (SAM Interfaces).
- As name suggest, they permit exactly **one abstract method** inside them.
- Java 8 introduces an optional annotation i.e. **@FunctionalInterface** which can be essentially used to capture Compile Time Errors.

```
@FunctionalInterface
interface IMathFunction {
    int calculate(int a, int b);
}
```

Lambda Expressions

BridgeLabz
Employability Delivered

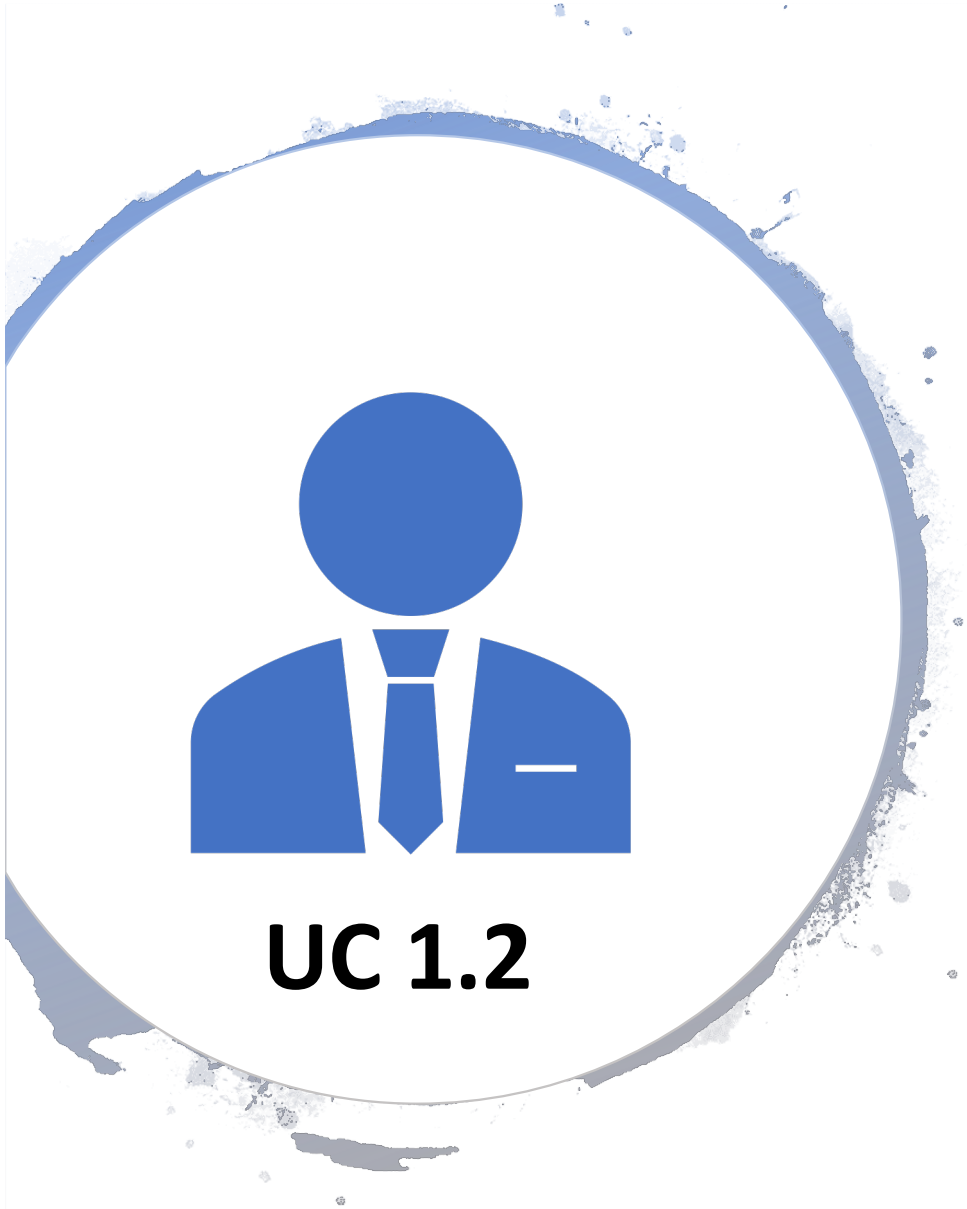
- A lambda expression is an **Anonymous Function**. A function that doesn't have a name and doesn't belong to any class.
- To create a lambda expression, we specify input parameters (if there are any) on the left side of the lambda operator ->, and place the expression or block of statements on the right side of lambda.

```
//Syntax of lambda expression
(parameter_list) -> {function_body}
```

```
(x, y) -> x * y;
```

- **Note:** Use of **Method Reference** to execute addition

```
// Using Method Reference instead of Lambda Expression
// This expression implements 'IMathFunction' interface
IMathFunction add = Integer::sum;

// Lambda expression for multiplication & division for two parameters.
// This expression implements 'IMathFunction'
IMathFunction multiply = (x, y) -> x * y;
IMathFunction divide = (int x, int y) -> x / y;

// Add & Multiply two numbers using lambda expression
System.out.println("Addition is " + add.calculate( a: 6,  b: 3));
System.out.println("Multiplication is " + multiply.calculate( a: 6,  b: 3));
System.out.println("Division is " + divide.calculate( a: 6,  b: 3));
```



Ability to show the results of Math Operation of Addition, Subtraction and Division

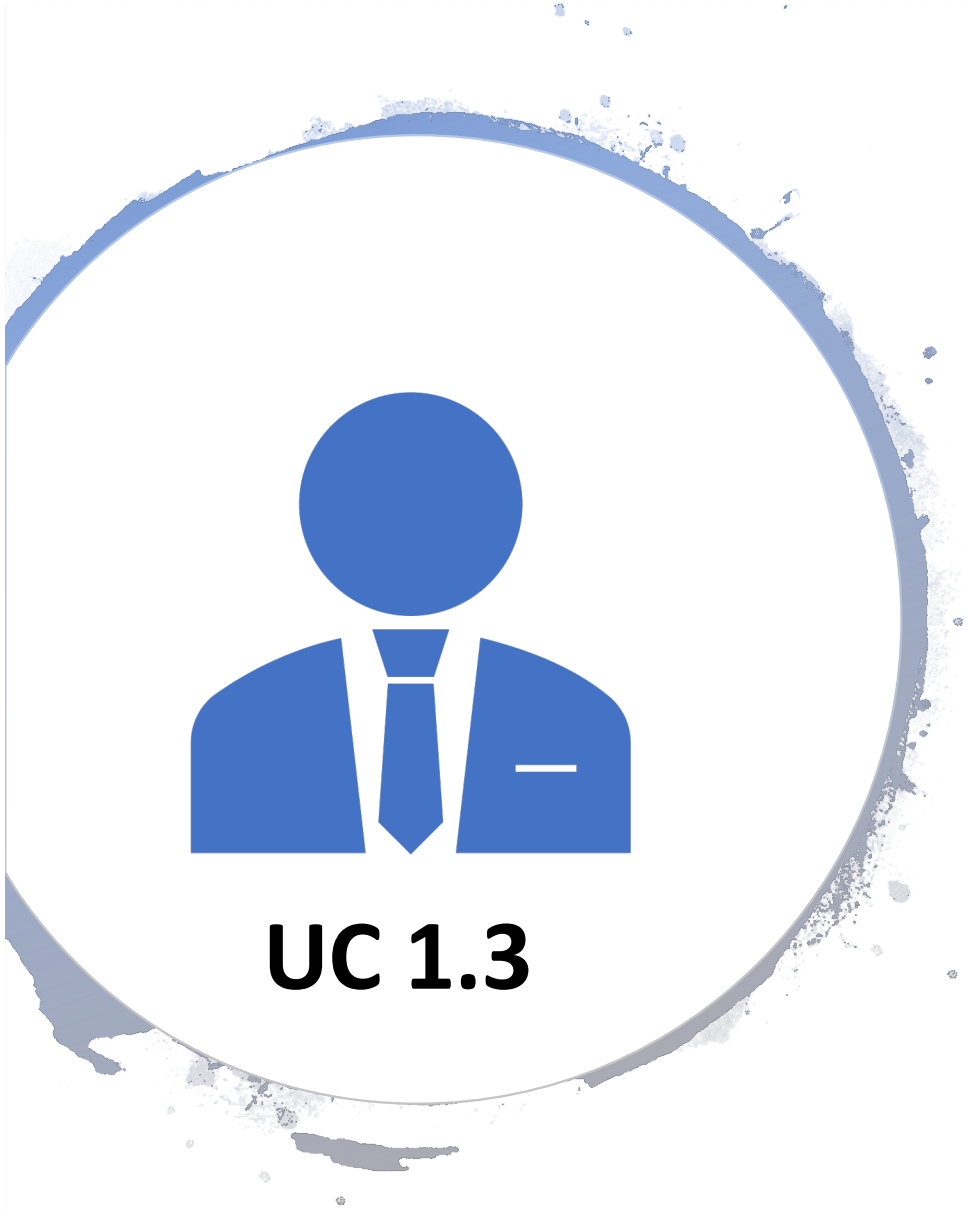
- Pass Lambda Function to show result

Lambda Expressions Motivations

BridgeLabz
Employability Delivered

- Reduced Line of Code rather than using a proper class or **Anonymous Class**.
- A function that can be created without belonging to any class.
- Enable to treat Lambda Expression as a method argument and execute on demand.
- **Note:** Here static method is used. Java8 has introduced **static and default methods** to provide static or default implementation so that all implementing classes do not change.

```
@FunctionalInterface
interface IMathFunction {
    int calculate(int a, int b);
    static void printResult(int a, int b, String function, IMathFunction fobj) {
        System.out.println("Result of "+function+" is "+fobj.calculate(a, b));
    }
}

// Passing Lambda Expression as Function Parameter to Print Result using Static Function
IMathFunction.printResult( a: 6,  b: 3,  function: "Addition", add);
IMathFunction.printResult( a: 6,  b: 3,  function: "Multiplication", multiply);
IMathFunction.printResult( a: 6,  b: 3,  function: "Division", divide);
```



Create a Number Play List and Iterate and print each element

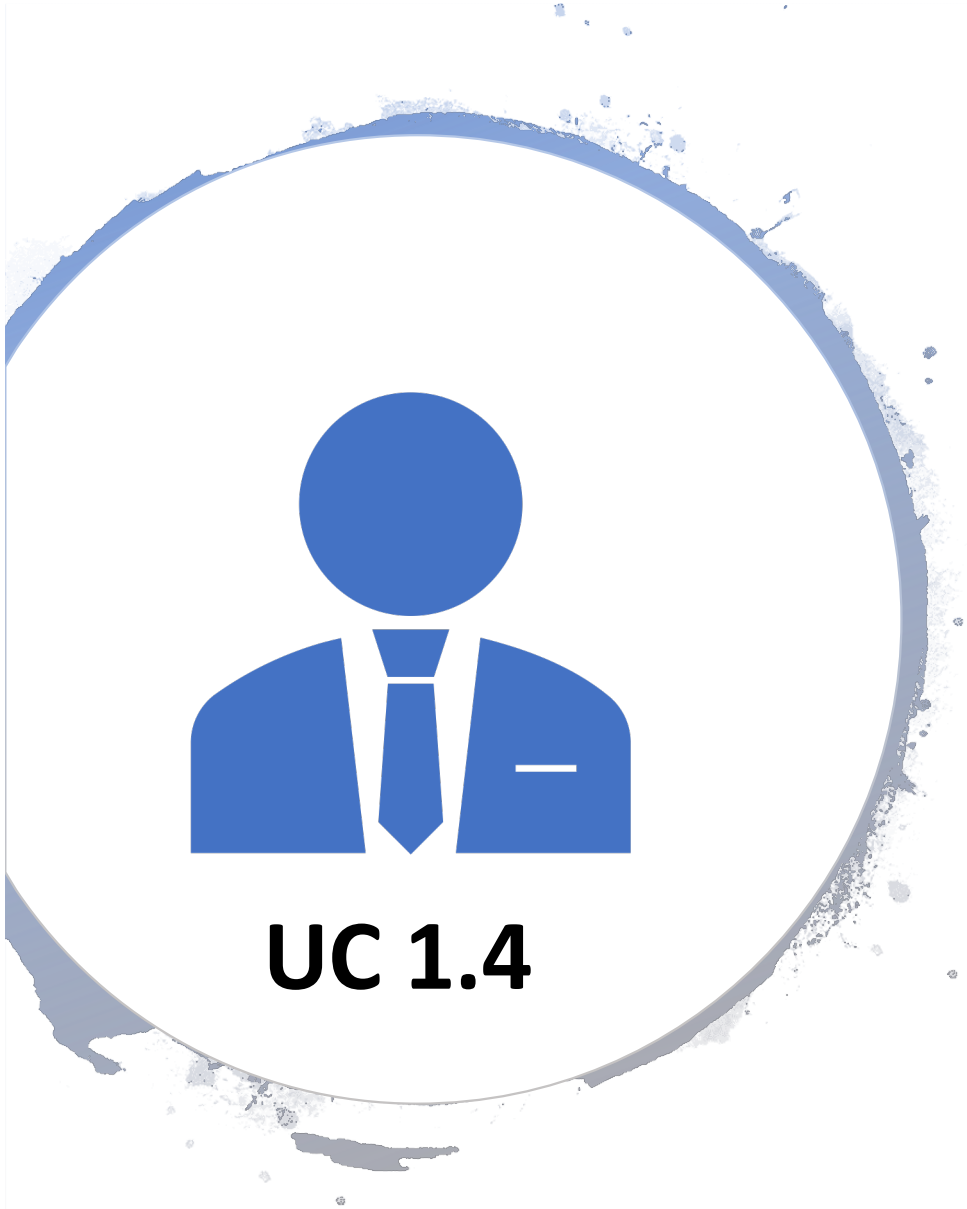
- Use forEach to demonstrate Iteration
- Show using proper Class, Anonymous Class and Lambda Function

forEach Method & Lambda Expressions

Java 8 has introduced **forEach** method which takes **java.util.function.Consumer** object as argument for iterating through each element of a Collection Object

```
//Method 1: traversing using Iterator
Iterator<Integer> it = myList.iterator();
while(it.hasNext()){
    Integer i = it.next();
    System.out.println("Iterator Value::"+i);
}

//Method 2: Traversing with Consumer interface implementation
class MyConsumer implements Consumer<Integer>{
    public void accept(Integer t) {
        System.out.println("Mth2: Consumer impl Value::"+t);
    }
}
MyConsumer action = new MyConsumer();
myList.forEach(action);
```

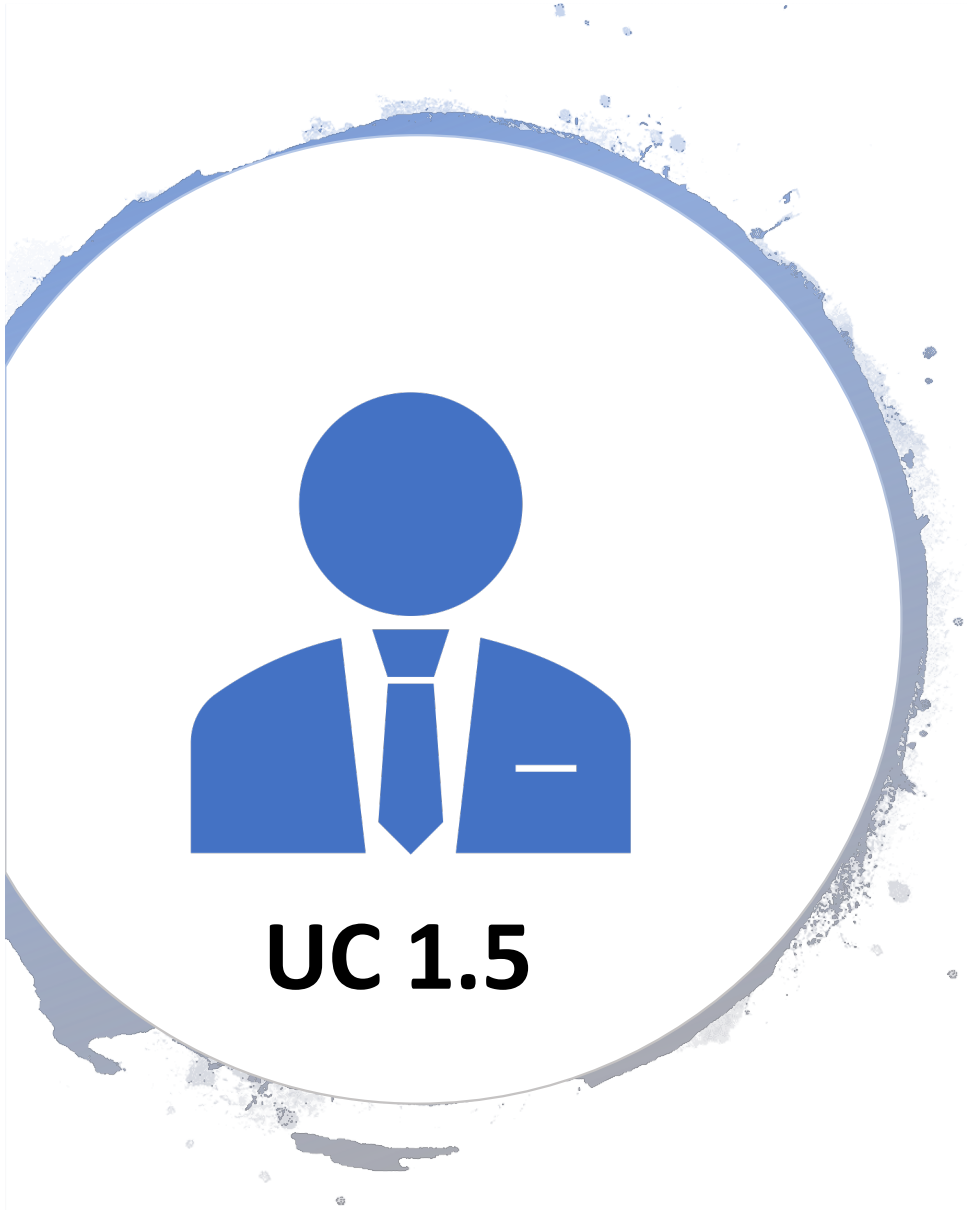
```
//Method 3: Traversing with Anonymous Consumer interface implementation
myList.forEach(new Consumer<Integer>() {
    public void accept(Integer t) {
        System.out.println("Mth3: forEach anonymous class Value::"+t);
    }
});

//Method 4: Explicit Lambda Function
Consumer<Integer> myListAction = n-> {
    System.out.println("Mth4: forEach Lambda impl Value::" + n);
};
myList.forEach(myListAction);

//Method 5: Implicit Lambda Function
myList.forEach(n-> {
    System.out.println("Mth5: forEach Lambda impl Value::" + n);
});
```



UC 1.4

Ability to Iterate through List of number and print each element in double

- Use function Functional Interface for conversion



Ability to Iterate through List of number and print only if it is even

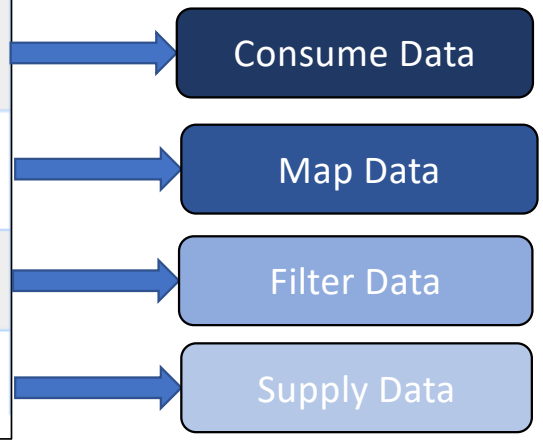
- Use predicate Functional Interface to check

Java Predefined Functional Interface

BridgeLabz
Employability Delivered

The **java.util.function** package defines several predefined functional interfaces that you can use when creating lambda expressions. Most of this functional interface are used in Java Stream API. Here are some of the most important ones:

Functional Interface	Abstract Method	Function descriptor	Description
Consumer<T>	accept(T t)	T -> void	Represents an operation that accepts a single input argument and returns no result.
Function<T, R>	apply(T t)	T -> R	Represents a function that accepts one argument and produces a result.
Predicate<T>	test(T t)	T -> boolean	Represents a predicate (boolean-valued function) of one argument.
Supplier<T>	get()	() -> T	Represents a supplier of results.



Java Predefined Functional Interface

BridgeLabz
Employability Delivered

```
//Method 6: Implicit Lambda Function to print double value
Function<Integer,Double> doubleFunction = Integer::doubleValue;
myList.forEach(n-> {
    System.out.println("Mth5: forEach Lambda double Value::" +
                       doubleFunction.apply(n));
});

//Method 7: Implicit Lambda Function to check even
Predicate<Integer> isEvenFunction = n -> n%2 == 0;
myList.forEach(n-> {
    System.out.println("Mth5: forEach value of: "+n+
                       " check for Even: " + isEvenFunction.test(n));
});
```

BridgeLabz

Employability Delivered

Thank You