Java 8

By Bhanu Pratap Singh

Topics Included

1) Lambda function.
2) Function.
3) Consumer.
4) BiPredicate
5) Bi Consumer.
6)

Lambda expressions

Lamda Expression help us to implement SAM interface and we can write code in functional style

What is SAM : Single Abstract Method

```
@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
```

By Bhanu Pratap Singh

https://www.youtube.com/user/MrBhanupratap29/playlists
https://www.facebook.com/learnbybhanupratap/
https://www.udemy.com/javabybhanu

Lambda expressions

- Java lambda expression is used to provide the implementation of functional interface.
  Function interface means Interface with one method.
- Lambda expression saves lot of code lines.
- Lambda expressions are very effective when we have to iterate, filter and extract data from collection.

By Bhanu Pratap Singh

Lambda expressions Syntax

```
() -> { }
(p) -> {  }
(p1,p2) -> {  }

{} = body of lambda expression
() = argument to lambda expression
```

By Bhanu Pratap Singh

Lambda expressions Example

```
package lamda;

@FunctionalInterface
public interface FunctionalInterfaceExample2 {

void sayMessage(String message);
}
```

```
FunctionalInterfaceExample2 message1 = (a) -> {
a.toUpperCase();
};


FunctionalInterfaceExample2 message2 = (a) -> {
System.out.println(a.toUpperCase());
};
```

By Bhanu Pratap Singh

Lambda expressions Example

```
FunctionalInterfaceExample2 message = (a) -> { a.toUpperCase() };

FunctionalInterfaceExample2 message = (a) -> {
System.out.println(a.toUpperCase())};
```

```
message = (a) -> a.toUpperCase();

message = (a) -> System.out.println(a.toUpperCase());
```

By Bhanu Pratap Singh

Lambda expressions Example

```
package lamda;


@FunctionalInterface
public interface FunctionalInterfaceExample {

int operation(int a, int b);
}
```

By Bhanu Pratap Singh

Lambda expressions Example

```
FunctionalInterfaceExample add1 = (a, b) -> {
return a + b;
};

FunctionalInterfaceExample myltiply1 = (a, b) -> {
return a * b;
};

FunctionalInterfaceExample substract1 = (a, b) -> {
return a - b;
};
```

By Bhanu Pratap Singh

Lambda expressions Example

```
FunctionalInterfaceExample add = (a, b) -> a + b;

FunctionalInterfaceExample myltiply = (a, b) -> a * b;

FunctionalInterfaceExample substract = (a, b) -> a - b;
```

By Bhanu Pratap Singh

Lambda expressions Example

```
package function;

import java.util.function.Function;

public class UpperCaseFunction {

    Function<String, String> upperCase = (name) -> name.toUpperCase();

    public static void main(String[] args) {

    UpperCaseFunction upperCaseFunction = new UpperCaseFunction();

        System.out.println(upperCaseFunction.upperCase.apply("test"));
    }
}
```

By Bhanu Pratap Singh

Lambda expressions Example

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

public class AddStringFunction {

    static Function<String, String> addString = (name) -> name.toUpperCase().concat("Ram");

    public static void main(String[] args) {

    String value = AddStringFunction.addString.apply("test");
        System.out.println(value); // TESTRam
    }

}
```

By Bhanu Pratap Singh

Lambda expressions Example

```
package function;

import java.util.function.Function;

public class StringLengthFunction {

    static Function<String, Integer> length = (name) -> name.length();

    public static void main(String[] args) {

        int value = StringLengthFunction.length.apply("test");

        System.out.println(value); // 4
    }
}
```

By Bhanu Pratap Singh

https://www.youtube.com/user/MrBhanupratap29/playlists
https://www.facebook.com/learnbybhanupratap/
https://www.udemy.com/javabybhanu

Lambda expressions Example

```
package function;

import java.util.function.Function;

    public class StringContainsFunction {

    static Function<String, Boolean> contains = (name) -> name.contains("Test");

    public static void main(String[] args) {
        System.out.println(StringContainsFunction.contains.apply("t")); // false
        System.out.println(StringContainsFunction.contains.apply("Test")); //
        false
    }
}
```

By Bhanu Pratap Singh

Lambda expressions Example

```
package function;

import java.util.function.Function;

public class FindGraterFunction {

    static Function<Integer, Boolean> grater = (age) -> (age > 30);

    public static void main(String[] args) {
        System.out.println(grater.apply(90)); // true
        System.out.println(grater.apply(10)); // false
    }
}
```

By Bhanu Pratap Singh

Lambda expressions Example

```
package function;


import java.util.function.Function;


public class FindPrimeFunction {

    static Function<Integer, Integer> isPrimeNumber = (number) -> (number % 2);
    public static void main(String[] args) {
        System.out.println(isPrimeNumber.apply(90)); //0
        System.out.println(isPrimeNumber.apply(11)); //1

    }
}
```

By Bhanu Pratap Singh

DoubleFunction

```
/**
 * Represents a function that accepts a double-valued argument and produces a result.
 * @param <R> the type of the result of the function
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface DoubleFunction<R> {


    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    R apply(double value);
}
```

By Bhanu Pratap Singh

DoubleFunction Example

```java
package function;

import java.util.function.DoubleFunction;

public class DoubleFunctionExample {

    static DoubleFunction<Integer> doubleFunction = d -> (int)(d * 10);

    public static void main(String[] args) {

        System.out.println(doubleFunction.apply(20.90)); // 290
    }
}
```

By Bhanu Pratap Singh

DoubleToIntFunction

```java
/**
 * Represents a function that accepts a double-valued argument and produces
an
 * int-valued result.
 *
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface DoubleToIntFunction {

    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    int applyAsInt(double value);
}
```

By Bhanu Pratap Singh

https://www.youtube.com/user/MrBhanupratap29/playlists
https://www.facebook.com/learnbybhanupratap/
https://www.udemy.com/javabybhanu

DoubleToIntFunction Example

```
package function;

import java.util.function.DoubleToIntFunction;

public class DoubleToIntFunctionExample1 {

    static DoubleToIntFunction doubleToIntFunction = fun -> (int) fun;

    public static void main(String[] args) {

       System.out.println(doubleToIntFunction.applyAsInt(90.47565476534));//
       90
     }
}
```

By Bhanu Pratap Singh

DoubleToLongFunction

```
package java.util.function;
/**
 * Represents a function that accepts a double-valued argument and produces
a
 * long-valued result.
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface DoubleToLongFunction {
    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    long applyAsLong(double value);
}
```

By Bhanu Pratap Singh

DoubleToLongFunction Example

```
package function;

import java.util.function.DoubleToLongFunction;

public class DoubleToLongFunctionExample {

    // long rang -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

    static DoubleToLongFunction doubleToLongFunction = fun -> (long) fun;

    public static void main(String[] args) {
        System.out.println(doubleToLongFunction.applyAsLong(98744434333232233232.647487
        0898666));
        // 9223372036854775807
        }
}
```

By Bhanu Pratap Singh

IntFunction

```
/**
 * Represents a function that accepts an int-valued argument and produces a
 * result.
 * @param <R> the type of the result of the function
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface IntFunction<R> {

    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    R apply(int value);
}
```

By Bhanu Pratap Singh

IntFunction Example

```java
package function;

import java.util.function.IntFunction;

public class IntFunctionExample {

    static IntFunction<Integer> intFunction = fun -> fun * fun;

    public static void main(String[] args) {

        System.out.println(intFunction.apply(90));// 8100
    }
}
```

By Bhanu Pratap Singh

IntToDoubleFunction

```java
package java.util.function;

/**
 * Represents a function that accepts an int-valued argument and produces a
 * double-valued result
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface IntToDoubleFunction {

    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    double applyAsDouble(int value);
}
```

By Bhanu Pratap Singh

IntToDoubleFunction Example

```java
package function;

import java.util.function.IntToDoubleFunction;

public class IntToDoubleFunctionExample {

    static IntToDoubleFunction intToDoubleFunction = fun -> fun * fun;

    public static void main(String[] args) {
        System.out.println(intToDoubleFunction.applyAsDouble(7648));//
        5.8491904E7
        System.out.println(intToDoubleFunction.applyAsDouble(7));// 49.0
    }
}
```

By Bhanu Pratap Singh

LongFunction

```java
package java.util.function;

/**
 * Represents a function that accepts a long-valued argument and produces a
 * result.
 * @param <R> the type of the result of the function
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface LongFunction<R> {
    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    R apply(long value);
}
```

By Bhanu Pratap Singh

LongFunction Example

```java
package function;

import java.util.function.LongFunction;

public class LongFunctionExample {

    static LongFunction<String> longFunction = fun -> String.valueOf(fun * 2);

    public static void main(String[] args) {
        String value = longFunction.apply(567544l);
        System.out.println(value);// 1135088
    }
}
```

By Bhanu Pratap Singh

LongToDoubleFunction

```java
package java.util.function;
/**
 * Represents a function that accepts a long-valued argument and produces a
 * double-valued result.
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface LongToDoubleFunction {
    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    double applyAsDouble(long value);
}
```

By Bhanu Pratap Singh

LongToDoubleFunction Example

```java
package function;

import java.util.function.LongToDoubleFunction;

public class LongToDoubleFunctionExample {

    static LongToDoubleFunction longToDoubleFunction = fun -> fun/9.0;
    public static void main(String[] args) {
        System.out.println(longToDoubleFunction.applyAsDouble(Long.MAX_VALUE));//
        1.0248191152060861 4E18
        System.out.println(longToDoubleFunction.applyAsDouble(71l));//7.888888888888889
    }
}
```

By Bhanu Pratap Singh

LongToIntFunction

```java
package java.util.function;

/**
 * Represents a function that accepts a long-valued argument and produces an
 * int-valued result.
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface LongToIntFunction {
    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    int applyAsInt(long value);
}
```

By Bhanu Pratap Singh

LongToIntFunction Example

```java
package function;

import java.util.function.LongToIntFunction;

public class LongToIntFunctionExample {

    static LongToIntFunction longToIntFunction = fun -> (int) (fun);
    public static void main(String[] args) {
        System.out.println(longToIntFunction.applyAsInt(440L)); //440
    }
}
```

By Bhanu Pratap Singh

ToDoubleBiFunction

```java
package java.util.function;
/**
 * Represents a function that accepts two arguments and produces a double-
valued
 * result.
 * @param <T> the type of the first argument to the function
 * @param <U> the type of the second argument to the function
 * @see BiFunction
 * @since 1.8
 */
@FunctionalInterface
public interface ToDoubleBiFunction<T, U> {
    /**
     * Applies this function to the given arguments.
     * @param t the first function argument
     * @param u the second function argument
     * @return the function result
     */
    double applyAsDouble(T t, U u);
}
```

By Bhanu Pratap Singh

https://www.youtube.com/user/MrBhanupratap29/playlists
https://www.facebook.com/learnbybhanupratap/
https://www.udemy.com/javabybhanu

ToDoubleBiFunction Example

```
package function;

import java.util.function.ToDoubleBiFunction;

public class ToDoubleBiFunctionExample {

    static ToDoubleBiFunction<Integer, Double> toDoubleBiFunction = (fun1, fun2) -> fun1 + fun2;
    static ToDoubleBiFunction<Integer, Integer> toDoubleBiFunction1 = (fun1, fun2) -> fun1*fun2;
    public static void main(String[] args) {
        System.out.println(toDoubleBiFunction.applyAsDouble(20, 70.678686));
        System.out.println(toDoubleBiFunction1.applyAsDouble(20, 70));
   }
}
90.678686
1400.0
```

By Bhanu Pratap Singh

ToDoubleFunction

```
package java.util.function;
/**
 * Represents a function that produces a double-valued result.
 * @param <T> the type of the input to the function
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface ToDoubleFunction<T> {
    /**
     * Applies this function to the given argument.
     *
     * @param value the function argument
     * @return the function result
     */
    double applyAsDouble(T value);
}
```

By Bhanu Pratap Singh

https://www.youtube.com/user/MrBhanupratap29/playlists
https://www.facebook.com/learnbybhanupratap/
https://www.udemy.com/javabybhanu

ToDoubleFunction Example

```
package function;

import java.util.function.ToDoubleFunction;

public class ToDoubleFunctionExample {

    static ToDoubleFunction<Integer> toDoubleFunction = fun1 -> fun1 * 301;

    public static void main(String[] args) {
        System.out.println(toDoubleFunction.applyAsDouble(90000)); // 2.709E7
        System.out.println(toDoubleFunction.applyAsDouble(900)); //270900.0
    }
}
```

By Bhanu Pratap Singh

ToIntBiFunction

```
package java.util.function;
/**
 * Represents a function that accepts two arguments and produces an int-
valued result
 * @param <T> the type of the first argument to the function
 * @param <U> the type of the second argument to the function
 * @see BiFunction
 * @since 1.8
 */
@FunctionalInterface
public interface ToIntBiFunction<T, U> {
    /**
     * Applies this function to the given arguments.
     * @param t the first function argument
     * @param u the second function argument
     * @return the function result
     */
    int applyAsInt(T t, U u);
}
```

By Bhanu Pratap Singh

ToIntBiFunction Example

```
package function;

import java.util.function.ToIntBiFunction;

public class ToIntBiFunctionExample {

     static ToIntBiFunction<String,String> toIntBiFunction  = (x,y)-> Integer.parseInt(x) +Integer.parseInt(4);

     public static void main(String[] args) {

          System.out.println(toIntBiFunction.applyAsInt("44","80")); // 124

          }

}
```

By Bhanu Pratap Singh

ToIntFunction

```
package java.util.function;
/**
 * Represents a function that produces an int-valued result.
 * @param <T> the type of the input to the function
 * @see Function
 * @since 1.8
 */
@FunctionalInterface
public interface ToIntFunction<T> {
    /**
     * Applies this function to the given argument.
     * @param value the function argument
     * @return the function result
     */
    int applyAsInt(T value);
}
```

By Bhanu Pratap Singh

https://www.youtube.com/user/MrBhanupratap29/playlists
https://www.facebook.com/learnbybhanupratap/
https://www.udemy.com/javabybhanu

ToLongBiFunction

```
package java.util.function;
/**
 * Represents a function that accepts two arguments and produces a long-
valued result.
 * @param <T> the type of the first argument to the function
 * @param <U> the type of the second argument to the function
 * @see BiFunction
 * @since 1.8
 */
@FunctionalInterface
public interface ToLongBiFunction<T, U> {
    /**
     * Applies this function to the given arguments.
     * @param t the first function argument
     * @param u the second function argument
     * @return the function result
     */
    long applyAsLong(T t, U u);
}
```

By Bhanu Pratap Singh

Function Example

```
ToIntFunction<Integer> toIntFunction = fun -> fun * 1000;


ToLongBiFunction<Integer, Integer> toLongBiFunction = (fun1, fun2) -> fun1 - fun2;


ToLongFunction<Integer> toLongFunction = fun1 -> fun1 + 5;
```

By Bhanu Pratap Singh

Consumer

Consumer is interface which came as part of java8. It is present in java.utill package.
It represents a function which takes in one argument and produces a result. However these kind of functions don't return any value.

```
package java.util.function;

import java.util.Objects;

/**
```

```
 * Represents an operation that accepts a single input argument and returns no
 * result.
 * @param <T> the type of the input to the operation
 *
 * @since 1.8
 */
@FunctionalInterface
public interface Consumer<T> {

    /**
     * Performs this operation on the given argument.
     *
     * @param t the input argument
     */
    void accept(T t);

    /**
     * Returns a composed {@code Consumer} that performs, in sequence, this
     * operation followed by the {@code after} operation. If performing either
     * operation throws an exception, it is relayed to the caller of the
     * composed operation.  If performing this operation throws an exception,
     * the {@code after} operation will not be performed.
     *
     * @param after the operation to perform after this operation
     * @return a composed {@code Consumer} that performs in sequence this
     * operation followed by the {@code after} operation
     * @throws NullPointerException if {@code after} is null
     */
    default Consumer<T> andThen(Consumer<? super T> after) {
        Objects.requireNonNull(after);
        return (T t) -> { accept(t); after.accept(t); };
    }
}
```

```
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample1 {

        public static void main(String[] args) {

                Consumer<Integer> con = p -> System.out.println(p);
```

```
		List<Integer> list = new ArrayList<Integer>();
		list.add(100);
		list.add(200);
		list.add(300);
		list.forEach(con);

	}
}
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample2 {

	public static void main(String[] args) {

		Consumer<Integer> con = p -> System.out.println(p*p);

		List<Integer> list = new ArrayList<Integer>();
		list.add(10);
		list.add(20);
		list.add(30);

		list.forEach(con);

	}
}
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample3 {

	public static void main(String[] args) {

		List<Integer> newList = new ArrayList<Integer>();

		Consumer<Integer> con = p -> {
			newList.add(p * 2);
		};
```

```
		Consumer<Integer> con1 = p -> System.out.println(p);

		List<Integer> list = new ArrayList<Integer>();
		list.add(10);
		list.add(20);
		list.add(30);

		list.forEach(con);

		newList.forEach(con1);

	}
}
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample4 {

	Consumer<Person> nameConsumer = person -> {
		if (person.getName().contains("Ram")) {
			person.setName(person.getName().toUpperCase());
			System.out.println(person);
		}
		else {
			System.out.println(person);
		}
	};

	Consumer<Person> ageConsumer = person -> {
		if (person.getAge() == 20) {
			person.setAge(person.getAge()+20);
			System.out.println(person);
		}
		else {
			System.out.println(person);
		}
	};

	public static void main(String[] args) {

		ConsumerExample4 example4 = new ConsumerExample4();
```

```
		List<Person> list = new ArrayList<Person>();
		list.add(new Person("Ram", 10));
		list.add(new Person("Mohan", 20));
		list.add(new Person("Sohan", 30));

		list.forEach(example4.nameConsumer);

		System.out.println("----------");
		list.forEach(example4.ageConsumer);

	}
}

class Person {

	private String name;
	private int age;

	public Person(String name, int age) {
		super();
		this.name = name;
		this.age = age;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	@Override
	public String toString() {
		return "Person [name=" + name + ", age=" + age + "]";
	}
```

```
}
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample5 {

	Consumer<List<Integer>> consumer = (list) -> {
		for (Integer li : list) {
			System.out.println(li);
		}
	};

	public static void main(String[] args) {

		ConsumerExample5 example4 = new ConsumerExample5();

		List<Integer> list = new ArrayList<Integer>();
		list.add(10);
		list.add(20);
		list.add(30);

		example4.consumer.accept(list);

	}

}
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample6 {

	Consumer<List<Employee>> consumer = (list) -> {
		for (Employee li : list) {
			if (li.getAge() > 19) {
				System.out.println(li);
			}
		}
	};
```

```
	public static void main(String[] args) {

		ConsumerExample6 example4 = new ConsumerExample6();

		List<Employee> list = new ArrayList<Employee>();
		list.add(new Employee("Test1", 10));
		list.add(new Employee("Test1", 20));
		list.add(new Employee("Test1", 30));

		example4.consumer.accept(list);

	}
}

class Employee {

	private String name;
	private int age;

	public Employee(String name, int age) {
		super();
		this.name = name;
		this.age = age;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	@Override
	public String toString() {
		return "Person [name=" + name + ", age=" + age + "]";
```

```
	}
}
package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerExample7 {

	static List<Integer> odd = new ArrayList<>();
	static List<Integer> even = new ArrayList<>();

	static Consumer<Integer> number = n -> {
		if (n % 2 == 0) {
			even.add(n);
		} else {
			odd.add(n);
		}
	};

	public static void main(String[] args) {

		Consumer<List<Integer>> printList = list -> list.forEach(n ->
System.out.println(n));

		number.accept(10);
		number.accept(15);
		number.accept(25);
		number.accept(30);

		printList.accept(odd);

		printList.accept(even);
	}
}
package consumer;

import java.util.function.Consumer;

public class ConsumerExample8 {

	Consumer<Citizen> consumer = c -> {
		if (c.getAge() < 18) {
			System.out.println(c.getName() + " not eligible to vote.");
```

```
                } else {
                        System.out.println(c.getName() + " eligible to vote.");
                }
        };


        public static void main(String[] args) {
                ConsumerExample8 electionConsumer = new ConsumerExample8();

                electionConsumer.consumer.accept(new Citizen("Ram", 15));

                electionConsumer.consumer.accept(new Citizen("Mohan", 20));
        }
}

class Citizen {
        private String name;
        private int age;

        public Citizen(String name, int age) {
                this.name = name;
                this.age = age;
        }

        public String getName() {
                return name;
        }

        public int getAge() {
                return age;
        }
}

package consumer;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerAndThen1 {

        public static void main(String[] args) {

                List<Integer> list = Arrays.asList(300, 400, 500, 600);
```

```
		Consumer<List<Integer>> consumer = listConsumer -> {
			for (int i = 0; i < list.size(); i++) {
				list.set(i, list.get(i) * list.get(i));
			}
		};

		Consumer<List<Integer>> printConsumer = listConsumer -> list.forEach(n ->
System.out.println(n));

		consumer.andThen(printConsumer).accept(list);
	}
}
package consumer;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerAndThen2 {

	public static void main(String[] args) {

		MyNumber myNumber = new MyNumber();

		List<Integer> list = Arrays.asList(8, 7, 12, 17, 14, 13);

		Consumer<List<Integer>> oddNumConsumer = myNumber::printOddNum;

		Consumer<List<Integer>> evenNumConsumer = myNumber::printEvenNum;

		Consumer<List<Integer>> doneConsumer = myNumber::done;


	oddNumConsumer.andThen(evenNumConsumer).andThen(doneConsumer).accept(li
st);
	}
}

class MyNumber {
	void printOddNum(List<Integer> myNumbers) {
		System.out.println("--printOddNum--");
		myNumbers.forEach(n -> {
			if (n % 2 == 1) {
				System.out.println(n + " ");
			}
```

```
		});
	}

	void printEvenNum(List<Integer> myNumbers) {
		System.out.println("--printEvenNum--");
		myNumbers.forEach(n -> {
			if (n % 2 == 0) {
				System.out.println(n + " ");
			}
		});
	}

	void done(List<Integer> myNumbers) {
		System.out.println("processign done");
	}
}

package consumer;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerAndThen4 {

	public static void main(String[] args) {

		List<Integer> list = Arrays.asList(300, 400, 500, 600);

		Consumer<List<Integer>> printConsumer =
				listConsumer -> list.forEach(System.out::println);

		printConsumer.accept(list);
	}
}

package consumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerAndThen5 {

	public static void main(String[] args) {
```

```
		List<Crona> list = new ArrayList<Crona>();
		list.add(new Crona("US", 1244302));
		list.add(new Crona("India", 50000));

		Consumer<List<Crona>> printCountryConsumer = listConsumer ->
list.forEach(Crona::getCountry);

		Consumer<List<Crona>> printCasesConsumer = listConsumer ->
list.forEach(Crona::getTotalCases);

		printCountryConsumer.andThen(printCasesConsumer).accept(list);

	}
}

class Crona {

	String country;
	Integer totalCases;

	public Crona(String country, Integer totalCases) {
		super();
		this.country = country;
		this.totalCases = totalCases;
	}

	public String getCountry() {
		System.out.println(country);
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}

	public Integer getTotalCases() {
		System.out.println(totalCases);
		return totalCases;
	}

	public void setTotalCases(Integer totalCases) {
		this.totalCases = totalCases;
	}
```

```
}
package consumer;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class ConsumerAndThen6 {

	static Consumer<List<Integer>> consumer1 = new Consumer<List<Integer>>() {
		@Override
		public void accept(List<Integer> t) {
			System.out.println("in consumer1");
			for (int i = 0; i < t.size(); i++) {
				if (t.get(i) % 2 == 0) {
					System.out.println(t.get(i));
				}
			}

		}
	};

	static Consumer<Integer> consumer2 = (Integer x) -> System.out.println(x);

	public static void main(String[] args) {

		List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 8, 10);

		forEach(list, consumer2);
		System.out.println("==consumer2 ends==");
		forEach(list, (Integer x) -> System.out.println(x));

		consumer1.accept(list);

	}

	static <T> void forEach(List<T> list, Consumer<T> consumer) {
		for (T t : list) {
			consumer.accept(t);
		}
	}

}

package java.util.function;
```

```java
import java.util.Objects;

/**
 * Represents an operation that accepts two input arguments and returns no
 * result.
* @param <T> the type of the first argument to the operation
 * @param <U> the type of the second argument to the operation
 * @see Consumer
 * @since 1.8
 */
@FunctionalInterface
public interface BiConsumer<T, U> {

    /**
     * Performs this operation on the given arguments.
     *
     * @param t the first input argument
     * @param u the second input argument
     */
    void accept(T t, U u);

    /**
     * Returns a composed {@code BiConsumer} that performs, in sequence, this
     * operation followed by the {@code after} operation. If performing either
     * operation throws an exception, it is relayed to the caller of the
     * composed operation.  If performing this operation throws an exception,
     * the {@code after} operation will not be performed.
     *
     * @param after the operation to perform after this operation
     * @return a composed {@code BiConsumer} that performs in sequence this
     * operation followed by the {@code after} operation
     * @throws NullPointerException if {@code after} is null
     */
    default BiConsumer<T, U> andThen(BiConsumer<? super T, ? super U> after) {
        Objects.requireNonNull(after);

        return (l, r) -> {
            accept(l, r);
            after.accept(l, r);
        };
    }
}

package biConsumer;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.function.BiConsumer;

public class BiConsumerExample3 {

	static Map<Integer, String> newMap = new HashMap<Integer, String>();

	static BiConsumer<Map<Integer, String>, Map<Integer, String>> mapConsumer =
(map1, map2) -> {
		for (Map.Entry<Integer, String> map : map1.entrySet()) {
			map2.put(map.getKey() * 3, map.getValue());
		}
	};

	public static void main(String args[]) {
		Map<Integer, String> map = new HashMap<>();

		map.put(1, "Java");
		map.put(2, "Go");
		map.put(3, "C");
		map.put(4, "JavaScript");
		map.put(5, "C++");

		mapConsumer.accept(map, newMap);

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

```
package biConsumer;

import java.util.ArrayList;
import java.util.List;
import java.util.function.BiConsumer;

public class BiConsumerExample5 {

	public static void main(String args[]) {

		List<Integer> list1 = new ArrayList<Integer>();
		list1.add(2);
		list1.add(1);
		list1.add(3);
```

```
		list1.add(5);

		List<Integer> list2 = new ArrayList<Integer>();
		list2.add(2);
		list2.add(1);
		list2.add(4);

		BiConsumer<List<Integer>, List<Integer>> compareBiConsumer = (li1, li2) -> {
			for (int i = 0; i < li1.size(); i++) {
				if (li1.get(i) != li2.get(i)) {
					System.out.println(li1.get(i) + " not equal");
				} else {
					System.out.println(li1.get(i) + " equal");
				}
			}

		};

		BiConsumer<List<Integer>, List<Integer>> display = (li1, li2) -> {
			li1.forEach(a -> System.out.print(a + " "));
			System.out.println();
			li2.forEach(a -> System.out.print(a + " "));
			System.out.println();
		};

		try {
			compareBiConsumer.andThen(display).accept(list1, list2);
		} catch (Exception e) {
			System.out.println("Exceptiion : " + e);
		}
	}
}
```

BiPredicate

```
package java.util.function;

import java.util.Objects;

/**
 * Represents a predicate (boolean-valued function) of two arguments.  This is
 * the two-arity specialization of {@link Predicate}.
 *
 * <p>This is a <a href="package-summary.html">functional interface</a>
 * whose functional method is {@link #test(Object, Object)}.
 *
 * @param <T> the type of the first argument to the predicate
```

```
 * @param <U> the type of the second argument the predicate
 *
 * @see Predicate
 * @since 1.8
 */
@FunctionalInterface
public interface BiPredicate<T, U> {

    /**
     * Evaluates this predicate on the given arguments.
     *
     * @param t the first input argument
     * @param u the second input argument
     * @return {@code true} if the input arguments match the predicate,
     * otherwise {@code false}
     */
    boolean test(T t, U u);

    /**
     * Returns a composed predicate that represents a short-circuiting logical
     * AND of this predicate and another.  When evaluating the composed
     * predicate, if this predicate is {@code false}, then the {@code other}
     * predicate is not evaluated.
     *
     * <p>Any exceptions thrown during evaluation of either predicate are relayed
     * to the caller; if evaluation of this predicate throws an exception, the
     * {@code other} predicate will not be evaluated.
     *
     * @param other a predicate that will be logically-ANDed with this
     *              predicate
     * @return a composed predicate that represents the short-circuiting logical
     * AND of this predicate and the {@code other} predicate
     * @throws NullPointerException if other is null
     */
    default BiPredicate<T, U> and(BiPredicate<? super T, ? super U> other) {
        Objects.requireNonNull(other);
        return (T t, U u) -> test(t, u) && other.test(t, u);
    }

    /**
     * Returns a predicate that represents the logical negation of this
     * predicate.
     *
     * @return a predicate that represents the logical negation of this
     * predicate
```

```
     */
    default BiPredicate<T, U> negate() {
        return (T t, U u) -> !test(t, u);
    }

    /**
     * Returns a composed predicate that represents a short-circuiting logical
     * OR of this predicate and another.  When evaluating the composed
     * predicate, if this predicate is {@code true}, then the {@code other}
     * predicate is not evaluated.
     *
     * <p>Any exceptions thrown during evaluation of either predicate are relayed
     * to the caller; if evaluation of this predicate throws an exception, the
     * {@code other} predicate will not be evaluated.
     *
     * @param other a predicate that will be logically-ORed with this
     *              predicate
     * @return a composed predicate that represents the short-circuiting logical
     * OR of this predicate and the {@code other} predicate
     * @throws NullPointerException if other is null
     */
    default BiPredicate<T, U> or(BiPredicate<? super T, ? super U> other) {
        Objects.requireNonNull(other);
        return (T t, U u) -> test(t, u) || other.test(t, u);
    }
}
```

```
package biPredicate;

import java.util.function.BiPredicate;

public class BiPredicateExample1 {

    public static void main(String[] args) {

        BiPredicate<String, Integer> filter = (x, y) -> {
            return x.length() == y;
        };

        boolean result = filter.test("hellojava", 9);
        System.out.println(result);  // true

        boolean result2 = filter.test("hellojava", 10);
        System.out.println(result2); // false
    }
```

```
}
package biPredicate;

import java.util.Arrays;
import java.util.List;
import java.util.function.BiPredicate;

public class BiPredicateExample2 {

	static BiPredicate<String, Integer> biPredicate = (name, score) -> {
		if (score > 80) {
			System.out.println(name + " is Topper");
			return true;
		}
		System.out.println(name + " is Not Topper");
		return false;
	};

	public static void main(String[] args) {

		List<Student> students = Arrays.asList(new Student("Test1", 90), new
Student("Test2", 94),
				new Student("Test3", 33), new Student("Test4", 2));

		biPredicate.test("Ram", 90);
		biPredicate.test("Mohan", 10);

		for (Student student : students) {
			biPredicate.test(student.getName(), student.getMarks());
		}

	}

}

class Student {

	String name;
	Integer marks;

	public Student(String name, int marks) {
		super();
		this.name = name;
		this.marks = marks;
```

```
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@Override
	public String toString() {
		return "Student [name=" + name + ", marks=" + marks + "]";
	}

	public Integer getMarks() {
		return marks;
	}

	public void setMarks(int marks) {
		this.marks = marks;
	}

}
package biPredicate;

import java.util.function.BiPredicate;

public class BiPredicateExample3 {

	public static void main(String[] args) {

		BiPredicate<Integer, String> condition = (i, s) -> i > 30 && s.startsWith("R");

		BiPredicate<Integer, String> condition1 = (i, s) -> i < 30 && s.startsWith("T");

		System.out.println(condition.test(20, "Test1"));
		System.out.println(condition.test(30, "RTest2"));
		System.out.println(condition.test(50, "Test3"));
		System.out.println("--------");
		System.out.println(condition.negate().test(20, "Test1"));
		System.out.println(condition.negate().test(30, "RTest2"));
		System.out.println(condition.negate().test(50, "Test3"));

		System.out.println("--------");
```

```
                System.out.println(condition.or(condition1).test(20, "Test1"));
                System.out.println(condition.or(condition1).test(30, "RTest2"));
                System.out.println(condition.or(condition1).test(50, "Test3"));

                System.out.println("--------");
                System.out.println(condition.and(condition1).test(20, "Test1"));
                System.out.println(condition.and(condition1).test(30, "RTest2"));
                System.out.println(condition.and(condition1).test(50, "Test3"));

        }
}
false
false
false
--------
true
true
true
--------
true
false
false
--------
false
false
false
```

BiFunction

```
package java.util.function;

import java.util.Objects;

/**
 * Represents a function that accepts two arguments and produces a result.
 * This is the two-arity specialization of {@link Function}.
 *
 * <p>This is a <a href="package-summary.html">functional interface</a>
 * whose functional method is {@link #apply(Object, Object)}.
 *
 * @param <T> the type of the first argument to the function
 * @param <U> the type of the second argument to the function
 * @param <R> the type of the result of the function
 *
 * @see Function
 * @since 1.8
```

```
 */
@FunctionalInterface
public interface BiFunction<T, U, R> {

    /**
     * Applies this function to the given arguments.
     *
     * @param t the first function argument
     * @param u the second function argument
     * @return the function result
     */
    R apply(T t, U u);

    /**
     * Returns a composed function that first applies this function to
     * its input, and then applies the {@code after} function to the result.
     * If evaluation of either function throws an exception, it is relayed to
     * the caller of the composed function.
     *
     * @param <V> the type of output of the {@code after} function, and of the
     *           composed function
     * @param after the function to apply after this function is applied
     * @return a composed function that first applies this function and then
     * applies the {@code after} function
     * @throws NullPointerException if after is null
     */
    default <V> BiFunction<T, U, V> andThen(Function<? super R, ? extends V> after) {
        Objects.requireNonNull(after);
        return (T t, U u) -> after.apply(apply(t, u));
    }
}

package biFunction;

import java.util.function.BiFunction;
import java.util.function.Function;

public class Example1 {

        static BiFunction<Integer, Integer, Integer> addFunction = (num1, num2) -> (num1 +
num2);

        static Function<Integer, Integer> multipleFunction = (num1) -> (num1 * 5);

        public static void main(String[] args) {
```

```
		System.out.println(addFunction.apply(10, 25));

		Integer status = addFunction.andThen(multipleFunction).apply(10, 25);
		System.out.println(status);
		status = addFunction.andThen(result -> result * 5).andThen(result -> result %
2).apply(10, 25);
		System.out.println(status);
	}
}
```

```
package biFunction;

import java.util.function.BiFunction;

public class Example2 {

	public static void main(String args[]) {
		BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;

		try {
			add = add.andThen(null);
			System.out.println("Add = " + add.apply(2, 3));
		} catch (Exception e) {
			System.out.println("Exception: " + e);
		}
	}

}
```

```
package biFunction;

import java.util.function.BiFunction;

public class Example3 {

	public static void main(String[] args) {

		BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;

		add = add.andThen(a -> a / 0);

		try {
			System.out.println("add = " + add.apply(2, 3));
		} catch (Exception e) {
```

```
                System.out.println("Exception: " + e);
            }
        }
}
```