(Java Roadmap)- (Instagram) - (Interview Preparation Website) - (Whatsapp)

100 LAMBDA's - Copy and run these programs to understand better

1. Simple Runnable Example

```
public class LambdaExample1 {
    public static void main(String[] args) {
        Runnable r = () -> System.out.println("Hello, Lambda!");
        new Thread(r).start();
    }
}
```

2. Iterating List using Lambda

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample2 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        list.forEach(item -> System.out.println(item));
    }
}
```

3. Filtering a List with Lambda

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample3 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "JavaScript", "Python");
        List<String> filteredList = list.stream()
                                        .filter(s -> s.startsWith("J"))
                                        .collect(Collectors.toList());
        filteredList.forEach(System.out::println);
    }
}
```

4. Sorting a List using Lambda

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample4 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Kafka");
        list.sort((s1, s2) -> s1.compareTo(s2));
        list.forEach(System.out::println);
    }
}
```

5. Using a Custom Functional Interface

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

```
public class LambdaExample5 {
    public static void main(String[] args) {
        Calculator add = (a, b) -> a + b;
        Calculator multiply = (a, b) -> a * b;


        System.out.println("Addition: " + add.calculate(5, 3));
        System.out.println("Multiplication: " + multiply.calculate(5, 3));
    }
}
```

6. Lambda with Map Iteration

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


public class LambdaExample6 {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("Java", 8);
        map.put("Spring", 5);
        map.put("Lambda", 1);


        map.forEach((key, value) -> System.out.println(key + ": " + value));
    }
}
```

7. Creating a Thread with Lambda

```
public class LambdaExample7 {
    public static void main(String[] args) {
        new Thread(() -> System.out.println("Thread with Lambda!")).start();
    }
}
```

8. Lambda in Comparator

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;


public class LambdaExample8 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Kafka");
        list.sort(Comparator.comparingInt(String::length));
        list.forEach(System.out::println);
    }
}
```

9. Method Reference with Lambda

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample9 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Kafka");
        list.forEach(System.out::println);
    }
}
```

10. Lambda with Optional

```
import java.util.Optional;
```

```
public class LambdaExample10 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.of("Java");
        optional.ifPresent(s -> System.out.println("Value is present: " + s));
    }
}
```

11. Lambda with Predicate

```
import java.util.function.Predicate;

public class LambdaExample11 {
    public static void main(String[] args) {
        Predicate<String> isEmpty = s -> s.isEmpty();
        System.out.println(isEmpty.test(""));   // true
        System.out.println(isEmpty.test("Java")); // false
    }
}
```

12. Lambda with BiFunction

```
import java.util.function.BiFunction;

public class LambdaExample12 {
    public static void main(String[] args) {
        BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;
        System.out.println(add.apply(2, 3));  // 5
    }
}
```

13. Lambda with Consumer

```
import java.util.function.Consumer;

public class LambdaExample13 {
    public static void main(String[] args) {
        Consumer<String> print = s -> System.out.println(s);
        print.accept("Hello, World!");  // Hello, World!
    }
}
```

14. Lambda with Supplier

```
import java.util.function.Supplier;

public class LambdaExample14 {
    public static void main(String[] args) {
        Supplier<String> supplier = () -> "Java";
        System.out.println(supplier.get());  // Java
    }
}
```

15. Lambda with Function

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

public class LambdaExample15 {
    public static void main(String[] args) {
        Function<String, Integer> length = s -> s.length();
        System.out.println(length.apply("Lambda"));  // 6
    }
```

```
}
```

16. Lambda with UnaryOperator

```
import java.util.function.UnaryOperator;

public class LambdaExample16 {
    public static void main(String[] args) {
        UnaryOperator<Integer> square = x -> x * x;
        System.out.println(square.apply(5));  // 25
    }
}
```

17. Lambda with BinaryOperator

```
import java.util.function.BinaryOperator;

public class LambdaExample17 {
    public static void main(String[] args) {
        BinaryOperator<Integer> multiply = (a, b) -> a * b;
        System.out.println(multiply.apply(2, 3));  // 6
    }
}
```

18. Lambda for Checking Even Numbers

```
import java.util.function.Predicate;

public class LambdaExample18 {
    public static void main(String[] args) {
        Predicate<Integer> isEven = x -> x % 2 == 0;
        System.out.println(isEven.test(4));  // true
        System.out.println(isEven.test(5));  // false
    }
}
```

19. Lambda with Custom Sorting

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample19 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Apple", "Banana", "Pear", "Grapes");
        list.sort((s1, s2) -> s2.compareTo(s1));  // Sort in reverse order
        list.forEach(System.out::println);
    }
}
```

20. Lambda for Uppercase Conversion

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class LambdaExample20 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("java", "spring", "lambda");
        List<String> upperList = list.stream()
                                     .map(String::toUpperCase)
                                     .collect(Collectors.toList());
        upperList.forEach(System.out::println);
    }
```

```
}
```

21. Lambda with Stream Reduce

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample21 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        int sum = numbers.stream()
                         .reduce(0, (a, b) -> a + b);
        System.out.println("Sum: " + sum);  // Sum: 15
    }
}
```

22. Lambda with Stream Filter

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample22 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        List<Integer> evenNumbers = numbers.stream()
                                           .filter(n -> n % 2 == 0)
                                           .collect(Collectors.toList());
        evenNumbers.forEach(System.out::println);  // 2, 4, 6, 8, 10
    }
}
```

23. Lambda with Stream Map

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample23 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        list.stream()
            .map(String::toLowerCase)
            .forEach(System.out::println);
    }
}
```

24. Lambda with Stream Distinct

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample24 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 2, 3, 4, 4, 5);
        numbers.stream()
               .distinct()
               .forEach(System.out::println);  // 1, 2, 3, 4, 5
    }
}
```

25. Lambda with Stream Sorted

```
import java.util.Arrays;
```

```
import java.util.List;

public class LambdaExample25 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Banana", "Apple", "Pear", "Grapes");
        list.stream()
            .sorted()
            .forEach(System.out::println);
    }
}
```

26. Lambda with Stream Count

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample26 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        long count = numbers.stream()
                            .count();
        System.out.println("Count: " + count);  // Count: 5
    }
}
```

27. Lambda with Stream AnyMatch

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample27 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        boolean containsJava = list.stream()
                                   .anyMatch(s -> s.equals("Java"));
        System.out.println("Contains 'Java': " + containsJava);  // true
    }
}
```

28. Lambda with Stream AllMatch

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample28 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(2, 4, 6, 8, 10);
        boolean allEven = numbers.stream()
                                 .allMatch(n -> n % 2 == 0);
        System.out.println("All even: " + allEven);  // true
    }
}
```

29. Lambda with Stream NoneMatch

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample29 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        boolean nonePython = list.stream()
```

```
                .noneMatch(s -> s.equals("Python"));
        System.out.println("Contains no 'Python': " + nonePython);  // true
    }
}
```

30. Lambda with Stream FindFirst

```
import java.util.Arrays;
import java.util.List;
import java.util.Optional;


public class LambdaExample30 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Optional<String> first = list.stream()
                                     .findFirst();
        first.ifPresent(System.out::println);  // Java
    }
}
```

31. Lambda with Stream FindAny

```
import java.util.Arrays;
import java.util.List;
import java.util.Optional;


public class LambdaExample31 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Optional<String> any = list.stream()
                                   .findAny();
        any.ifPresent(System.out::println);
    }
}
```

32. Lambda for Summing Integers

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample32 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        int sum = numbers.stream()
                         .mapToInt(Integer::intValue)
                         .sum();
        System.out.println("Sum: " + sum);  // Sum: 15
    }
}
```

33. Lambda for Averaging Integers

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample33 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        double average = numbers.stream()
                                .mapToInt(Integer::intValue)
                                .average()
                                .orElse(0.0);
        System.out.println("Average: " + average);  // Average: 3.0
```

```
    }
}
```

34. Lambda for Max Integer

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample34 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        int max = numbers.stream()
                        .mapToInt(Integer::intValue)
                        .max()
                        .orElse(Integer.MIN_VALUE);
        System.out.println("Max: " + max);  // Max: 5
    }
}
```

35. Lambda for Min Integer

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample35 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        int min = numbers.stream()
                        .mapToInt(Integer::intValue)
                        .min()
                        .orElse(Integer.MAX_VALUE);
        System.out.println("Min: " + min);  // Min: 1
    }
}
```

36. Lambda for Joining Strings

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample36 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String joined = list.stream()
                            .collect(Collectors.joining(", "));
        System.out.println(joined);  // Java, Spring, Lambda
    }
}
```

37. Lambda with Stream MapToInt

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample37 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        list.stream()
            .mapToInt(String::length)
            .forEach(System.out::println);  // 4, 6, 6
    }
}
```

38. Lambda with Stream Collect to Set

```
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;


public class LambdaExample38 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Spring");
        Set<String> set = list.stream()
                              .collect(Collectors.toSet());
        set.forEach(System.out::println);  // Java, Spring, Lambda
    }
}
```

39. Lambda with Stream GroupingBy

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample39 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Java");
        Map<String, Long> frequency = list.stream()
                                          .collect(Collectors.groupingBy(s -> s, Collectors.counting()));
        frequency.forEach((k, v) -> System.out.println(k + ": " + v));  // Java: 2, Spring: 1, Lambda: 1
    }
}
```

40. Lambda with Stream PartitioningBy

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample40 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        Map<Boolean, List<Integer>> partitioned = numbers.stream()
                                                         .collect(Collectors.partitioningBy(n -> n % 2 == 0));
        partitioned.forEach((k, v) -> System.out.println(k + ": " + v));  // true: [2, 4, 6, 8, 10], false: [1, 3, 5, 7, 9]
    }
}
```

41. Lambda with Stream Counting

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample41 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        long count = list.stream()
                         .collect(Collectors.counting());
        System.out.println("Count: " + count);  // Count: 3
    }
}
```

42. Lambda with Stream SummarizingInt

```
import java.util.Arrays;
import java.util.IntSummaryStatistics;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample42 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        IntSummaryStatistics stats = numbers.stream()
                                    .collect(Collectors.summarizingInt(Integer::intValue));
        System.out.println("Sum: " + stats.getSum());
        System.out.println("Average: " + stats.getAverage());
        System.out.println("Max: " + stats.getMax());
        System.out.println("Min: " + stats.getMin());
    }
}
```

43. Lambda with Stream Mapping

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample43 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Map<Integer, List<String>> map = list.stream()
                                    .collect(Collectors.groupingBy(String::length));
        map.forEach((k, v) -> System.out.println(k + ": " + v));  // 4: [Java], 6: [Spring, Lambda]
    }
}
```

44. Lambda with Stream Joining Without Delimiter

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample44 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String joined = list.stream()
                        .collect(Collectors.joining());
        System.out.println(joined);  // JavaSpringLambda
    }
}
```

45. Lambda with Stream ToMap

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample45 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Map<String, Integer> map = list.stream()
                                    .collect(Collectors.toMap(s -> s, String::length));
```

```
        map.forEach((k, v) -> System.out.println(k + ": " + v));  // Java: 4, Spring: 6, Lambda: 6
    }
}
```

46. Lambda for Creating a Stream

```
import java.util.stream.Stream;

public class LambdaExample46 {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("Java", "Spring", "Lambda");
        stream.forEach(System.out::println);
    }
}
```

47. Lambda with Stream Limit

```
import java.util.stream.Stream;

public class LambdaExample47 {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka");
        stream.limit(2)
              .forEach(System.out::println);  // Java, Spring
    }
}
```

48. Lambda with Stream Skip

```
import java.util.stream.Stream;

public class LambdaExample48 {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka");
        stream.skip(2)
              .forEach(System.out::println);  // Lambda, Kafka
    }
}
```

49. Lambda with Stream Peek

```
import java.util.stream.Stream;
import java.util.stream.Collectors;

public class LambdaExample49 {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka");
        stream.peek(System.out::println)
              .collect(Collectors.toList());
    }
}
```

50. Lambda with Optional

```
import java.util.Optional;

public class LambdaExample50 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.of("Java");
        optional.ifPresent(System.out::println);  // Java
    }
}
```

51. Lambda with Optional OrElse

```
import java.util.Optional;

public class LambdaExample51 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.ofNullable(null);
        String value = optional.orElse("Default");
        System.out.println(value);  // Default
    }
}
```

52. Lambda with Optional OrElseGet

```
import java.util.Optional;

public class LambdaExample52 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.ofNullable(null);
        String value = optional.orElseGet(() -> "Default");
        System.out.println(value);  // Default
    }
}
```

53. Lambda with Optional OrElseThrow

```
import java.util.Optional;

public class LambdaExample53 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.ofNullable(null);
        try {
            String value = optional.orElseThrow(() -> new RuntimeException("No value present"));
        } catch (Exception e) {
            System.out.println(e.getMessage());  // No value present
        }
    }
}
```

54. Lambda with Optional Map

```
import java.util.Optional;

public class LambdaExample54 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.of("Java");
        Optional<Integer> length = optional.map(String::length);
        length.ifPresent(System.out::println);  // 4
    }
}
```

55. Lambda with Optional Filter

```
import java.util.Optional;

public class LambdaExample55 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.of("Java");
        optional.filter(s -> s.equals("Java"))
                .ifPresent(System.out::println);  // Java
    }
}
```

56. Lambda with Optional FlatMap

```
import java.util.Optional;

public class LambdaExample56 {
    public static void main(String[] args) {
        Optional<String> optional = Optional.of("Java");
        optional.flatMap(s -> Optional.of(s.toUpperCase()))
                .ifPresent(System.out::println);  // JAVA
    }
}
```

57. Lambda for Custom Functional Interface

```
@FunctionalInterface
interface MyFunctionalInterface {
    void myMethod();
}

public class LambdaExample57 {
    public static void main(String[] args) {
        MyFunctionalInterface myFunc = () -> System.out.println("My method implementation");
        myFunc.myMethod();  // My method implementation
    }
}
```

58. Lambda for Custom Functional Interface with Parameter

```
@FunctionalInterface
interface MyFunctionalInterface {
    void myMethod(String s);
}

public class LambdaExample58 {
    public static void main(String[] args) {
        MyFunctionalInterface myFunc = (s) -> System.out.println(s);
        myFunc.myMethod("Hello, World!");  // Hello, World!
    }
}
```

59. Lambda for Custom Functional Interface with Return Value

```
@FunctionalInterface
interface MyFunctionalInterface {
    int myMethod(int a, int b);
}

public class LambdaExample59 {
    public static void main(String[] args) {
        MyFunctionalInterface add = (a, b) -> a + b;
        System.out.println(add.myMethod(5, 3));  // 8
    }
}
```

60. Lambda with Runnable

```
public class LambdaExample60 {
    public static void main(String[] args) {
        Runnable task = () -> System.out.println("Task is running");
        new Thread(task).start();  // Task is running
    }
}
```

61. Lambda with Comparator

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample61 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Pear", "Apple", "Banana");
        list.sort((s1, s2) -> s1.compareTo(s2));
        list.forEach(System.out::println);  // Apple, Banana, Pear
    }
}
```

62. Lambda with ForEach Loop

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample62 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        list.forEach(s -> System.out.println(s));
    }
}
```

63. Lambda for Array Sorting

```
import java.util.Arrays;


public class LambdaExample63 {
    public static void main(String[] args) {
        String[] array = {"Java", "Spring", "Lambda"};
        Arrays.sort(array, (s1, s2) -> s1.compareTo(s2));
        for (String s : array) {
            System.out.println(s);  // Java, Lambda, Spring
        }
    }
}
```

64. Lambda for Creating Threads

```
public class LambdaExample64 {
    public static void main(String[] args) {
        Runnable task1 = () -> System.out.println("Task 1 is running");
        Runnable task2 = () -> System.out.println("Task 2 is running");


        new Thread(task1).start();
        new Thread(task2).start();
    }
}
```

65. Lambda with Stream Reduce for Concatenation

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample65 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String result = list.stream()
                            .reduce("", (a, b) -> a + b);
        System.out.println(result);  // JavaSpringLambda
```

```
    }
}
```

66. Lambda with Stream Filter and Collect

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class LambdaExample66 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        List<String> filteredList = list.stream()
                                        .filter(s -> s.startsWith("S"))
                                        .collect(Collectors.toList());
        filteredList.forEach(System.out::println);  // Spring
    }
}
```

67. Lambda with Stream Map and Collect

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class LambdaExample67 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        List<String> upperList = list.stream()
                                     .map(String::toUpperCase)
                                     .collect(Collectors.toList());
        upperList.forEach(System.out::println);  // JAVA, SPRING, LAMBDA
    }
}
```

68. Lambda with Stream Filter and Count

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample68 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        long count = list.stream()
                         .filter(s -> s.contains("a"))
                         .count();
        System.out.println("Count: " + count);  // Count: 3
    }
}
```

69. Lambda with Stream Max

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class LambdaExample69 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String max = list.stream()
                         .max(Comparator.comparingInt(String::length))
                         .orElse("No max");
        System.out.println(max);  // Spring
```

```
    }
}
```

70. Lambda with Stream Min

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class LambdaExample70 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String min = list.stream()
                         .min(Comparator.comparingInt(String::length))
                         .orElse("No min");
        System.out.println(min);  // Java
    }
}
```

71. Lambda with Stream Collect to Map

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class LambdaExample71 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Map<String, Integer> map = list.stream()
                                       .collect(Collectors.toMap(s -> s, String::length));
        map.forEach((k, v) -> System.out.println(k + ": " + v));  // Java: 4, Spring: 6, Lambda: 6
    }
}
```

72. Lambda with Stream Map and FlatMap

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class LambdaExample72 {
    public static void main(String[] args) {
        List<List<String>> listOfLists = Arrays.asList(
            Arrays.asList("Java", "Spring"),
            Arrays.asList("Lambda", "Stream")
        );
        List<String> flatList = listOfLists.stream()
                                           .flatMap(List::stream)
                                           .collect(Collectors.toList());
        flatList.forEach(System.out::println);  // Java, Spring, Lambda, Stream
    }
}
```

73. Lambda with Stream Map and Reduce

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample73 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String concatenated = list.stream()
```

```
                .map(String::toUpperCase)
                .reduce("", (a, b) -> a + b);
        System.out.println(concatenated);  // JAVASPRINGLAMBDA
    }
}
```

74. Lambda with Stream Collect and GroupingBy

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class LambdaExample74 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Stream");
        Map<Integer, List<String>> grouped = list.stream()
                                        .collect(Collectors.groupingBy(String::length));
        grouped.forEach((k, v) -> System.out.println(k + ": " + v));  // 4: [Java], 6: [Spring, Lambda], 6: [Stream]
    }
}
```

75. Lambda with Stream Map and Counting

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample75 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        long count = list.stream()
                        .map(String::toUpperCase)
                        .count();
        System.out.println("Count: " + count);  // Count: 3
    }
}
```

76. Lambda with Stream Filter and FindFirst

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample76 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String first = list.stream()
                        .filter(s -> s.startsWith("S"))
                        .findFirst()
                        .orElse("Not found");
        System.out.println(first);  // Spring
    }
}
```

77. Lambda with Stream Collect and Joining

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class LambdaExample77 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String result = list.stream()
```

```
                .collect(Collectors.joining(", "));
        System.out.println(result);  // Java, Spring, Lambda
    }
}
```

78. Lambda with Stream AnyMatch

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample78 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        boolean anyMatch = list.stream()
                .anyMatch(s -> s.startsWith("S"));
        System.out.println(anyMatch);  // true
    }
}
```

79. Lambda with Stream AllMatch

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample79 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        boolean allMatch = list.stream()
                .allMatch(s -> s.length() > 3);
        System.out.println(allMatch);  // true
    }
}
```

80. Lambda with Stream NoneMatch

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample80 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        boolean noneMatch = list.stream()
                .noneMatch(s -> s.startsWith("Z"));
        System.out.println(noneMatch);  // true
    }
}
```

81. Lambda with Stream Sorted

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample81 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Spring", "Lambda", "Java");
        List<String> sortedList = list.stream()
                .sorted()
                .collect(Collectors.toList());
        sortedList.forEach(System.out::println);  // Java, Lambda, Spring
    }
}
```

82. Lambda with Stream Distinct

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample82 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Java", "Lambda");
        List<String> distinctList = list.stream()
                                        .distinct()
                                        .collect(Collectors.toList());
        distinctList.forEach(System.out::println);  // Java, Spring, Lambda
    }
}
```

83. Lambda with Stream Limit

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample83 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Stream");
        List<String> limitedList = list.stream()
                                       .limit(2)
                                       .collect(Collectors.toList());
        limitedList.forEach(System.out::println);  // Java, Spring
    }
}
```

84. Lambda with Stream Skip

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample84 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Stream");
        List<String> skippedList = list.stream()
                                       .skip(2)
                                       .collect(Collectors.toList());
        skippedList.forEach(System.out::println);  // Lambda, Stream
    }
}
```

85. Lambda with Stream Generate

```
import java.util.stream.Stream;

public class LambdaExample85 {
    public static void main(String[] args) {
        Stream.generate(() -> "Java")
              .limit(3)
              .forEach(System.out::println);  // Java, Java, Java
    }
}
```

86. Lambda with Stream Iterate

```
import java.util.stream.Stream;

public class LambdaExample86 {
    public static void main(String[] args) {
        Stream.iterate(1, n -> n + 2)
```

```
            .limit(5)
            .forEach(System.out::println);  // 1, 3, 5, 7, 9
    }
}
```

87. Lambda with Stream Collect to Unmodifiable List

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;


public class LambdaExample87 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        List<String> unmodifiableList = list.stream()
                                    .collect(Collectors.collectingAndThen(Collectors.toList(),
                                        collected -> List.copyOf(collected)));
        unmodifiableList.forEach(System.out::println);  // Java, Spring, Lambda
    }
}
```

88. Lambda with Stream forEachOrdered

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample88 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        list.stream()
            .forEachOrdered(System.out::println);  // Java, Spring, Lambda
    }
}
```

89. Lambda with Stream Reduce with Identity

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample89 {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
        int sum = list.stream()
                    .reduce(0, (a, b) -> a + b);
        System.out.println("Sum: " + sum);  // Sum: 15
    }
}
```

90. Lambda with Stream Reduce for Multiplication

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample90 {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 2, 3, 4);
        int product = list.stream()
                        .reduce(1, (a, b) -> a * b);
        System.out.println("Product: " + product);  // Product: 24
    }
}
```

91. Lambda with Stream Collect to Set

```
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public class LambdaExample91 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Java", "Lambda");
        Set<String> set = list.stream()
                              .collect(Collectors.toSet());
        set.forEach(System.out::println);  // Java, Spring, Lambda
    }
}
```

92. Lambda with Stream Count on Distinct Elements

```
import java.util.Arrays;
import java.util.List;

public class LambdaExample92 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Java", "Lambda");
        long count = list.stream()
                         .distinct()
                         .count();
        System.out.println("Count: " + count);  // Count: 3
    }
}
```

93. Lambda with Stream Filter for Empty Strings

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class LambdaExample93 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "", "Spring", "Lambda", "");
        List<String> nonEmptyList = list.stream()
                                        .filter(s -> !s.isEmpty())
                                        .collect(Collectors.toList());
        nonEmptyList.forEach(System.out::println);  // Java, Spring, Lambda
    }
}
```

94. Lambda with Stream Map for Object Transformation

```
import java.util.Arrays;
import java.util.List;

class Person {
    String name;
    int age;

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

public class LambdaExample94 {
```

```
    public static void main(String[] args) {
        List<String> names = Arrays.asList("John", "Jane", "Jack");
        List<Person> people = names.stream()
                                   .map(name -> new Person(name, 25))
                                   .collect(Collectors.toList());
        people.forEach(person -> System.out.println(person.name + ": " + person.age));  // John: 25, Jane: 25, Jack:
25
    }
}
```

95. Lambda with Stream Peek

```
import java.util.Arrays;
import java.util.List;


public class LambdaExample95 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        list.stream()
            .peek(System.out::println)
            .map(String::toUpperCase)
            .collect(Collectors.toList());
    }
}
```

96. Lambda with Stream Grouping and Counting

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample96 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Stream");
        Map<Integer, Long> groupedCount = list.stream()
                                              .collect(Collectors.groupingBy(String::length,
                                                  Collectors.counting()));
        groupedCount.forEach((k, v) -> System.out.println(k + ": " + v));  // 4: 1, 6: 3
    }
}
```

97. Lambda with Stream Collect to Map

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample97 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Map<Integer, String> map = list.stream()
                                       .collect(Collectors.toMap(String::length, s -> s));
        map.forEach((k, v) -> System.out.println(k + ": " + v));  // 4: Java, 6: Lambda (Spring might be overridden)
    }
}
```

98. Lambda with Stream Map to Optional

```
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
```

```
public class LambdaExample98 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        Optional<String> optional = list.stream()
                                        .filter(s -> s.startsWith("L"))
                                        .map(String::toUpperCase)
                                        .findFirst();
        optional.ifPresent(System.out::println);  // LAMBDA
    }
}
```

99. Lambda with Stream PartitioningBy

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample99 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda", "Stream");
        Map<Boolean, List<String>> partitioned = list.stream()
                                                     .collect(Collectors.partitioningBy(s -> s.length() > 4));
        partitioned.forEach((k, v) -> System.out.println(k + ": " + v));  // true: [Spring, Lambda, Stream], false: [Java]
    }
}
```

100. Lambda with Stream Min by Comparator

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;


public class LambdaExample100 {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "Spring", "Lambda");
        String shortest = list.stream()
                              .min(Comparator.comparingInt(String::length))
                              .orElse("Not found");
        System.out.println(shortest);  // Java
    }
}
```