100 LAMBDA CODE SNIPPETS

Java 8

1. Simple Runnable Example

```
Runnable r = () -> System.out.println("Hello, Lambda!");
new Thread(r).start();
```

2. Iterating List using Lambda

```
List<String> list = Arrays.asList("Java", "Spring", "Lambda");
list.forEach(item -> System.out.println(item));
```

3. Filtering a List with Lambda

```
List<String> list = Arrays.asList("Java", "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

```
List<String> list = Arrays.asList("Java", "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 static void main(String[] args) {
    Calculator add = (a, b) -> a + b;
    Calculator multiply = (a, b) -> a * b;
    println("Addition: " + add.calculate(5, 3));
    println("Multiplication: " + multiply.calculate(5, 3))
}
```

6. Lambda with Map Iteration

```
Map<String, Integer> map = new HashMap<>();
map.put("Java", 8);
map.put("Spring", 5);
map.put("Lambda", 1);
map.forEach((key, value) -> println(key + ": " + value));
```

7. Creating a Thread with Lambda

```
new Thread(() -> System.out.println("Thread with Lambda!")).start();
```

8. Lambda in Comparator

```
List<String> list = Arrays.asList("Java", "Lambda", "Kafka");
list.sort(Comparator.comparingInt(String::length));
list.forEach(System.out::println);
```

9. Method Reference with Lambda

```
List<String> list = Arrays.asList("Java", "Lambda", "Kafka");
list.forEach(System.out::println);
```

10. Lambda with Optional

```
Optional<String> optional = Optional.of("Java");
optional.ifPresent(s -> System.out.println("Value is present: " + s));
```

Sorting

```
myList.sort((s1,s2) -> s1.compareTo(s2));
myList.sort((s1,s2) -> Integer.compare(s1,s2));
myList.sort(Comparator.naturalOrder());
myList.sort(Comparator.comparingInt(a));
```

11. Lambda with Predicate

```
Predicate<String> isEmpty = s -> s.isEmpty();
System.out.println(isEmpty.test(""));    // true
System.out.println(isEmpty.test("Java")); // false
```

12. Lambda with BiFunction

```
BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;
System.out.println(add.apply(2, 3));  // 5
```

13. Lambda with Consumer

```
Consumer<String> print = s -> System.out.println(s)
print.accept("Hello, World!");  // Hello, World!
```

14. Lambda with Supplier

```
Supplier<String> supplier = () -> "Java";
System.out.println(supplier.get());  // Java
```

15. Lambda with Function

```
Function<String, Integer> length = s -> s.length();
System.out.println(length.apply("Lambda"));  // 6
```

16. Lambda with UnaryOperator

```
UnaryOperator<Integer> square = x -> x * x;
System.out.println(square.apply(5));  // 25
```

17. Lambda with BinaryOperator

```
BinaryOperator<Integer> multiply = (a, b) -> a * b;
System.out.println(multiply.apply(2, 3));  // 6
```

18. Lambda for Checking Even Numbers

```
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

```
List<String> list = Arrays.asList("Apple", "Pear", "Grapes");
list.sort((s1, s2) -> s2.compareTo(s1));// Sort, reverse order
list.forEach(System.out::println);
```

20. Lambda for Uppercase Conversion

```
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

```
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

```
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

```
List<String> list = Arrays.asList("Java", "Spring", "Lambda");
list.stream()
    .map(String::toLowerCase)
    .forEach(System.out::println);
```

24. Lambda with Stream Distinct

```
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

```
List<String> list = Arrays.asList("Banana", "Pear", "Grapes")
list.stream()
    .sorted()
    .forEach(System.out::println);
```

26. Lambda with Stream Count

```
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

```
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

```
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

```
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

```
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

```
List<String> list = Arrays.asList("Java", "Spring", "Lambda")
Optional<String> any = list.stream()
                           .findAny();
any.ifPresent(System.out::println);
```

32. Lambda for Summing Integers

```
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
```

34. Lambda for Max Integer

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda")
stream.forEach(System.out::println);
```

47. Lambda with Stream Limit

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka")
stream.limit(2)
      .forEach(System.out::println);  // Java, Spring
```

48. Lambda with Stream Skip

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka")
stream.skip(2)
      .forEach(System.out::println);  // Lambda, Kafka
```

49. Lambda with Stream Peek

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka")
stream.peek(System.out::println)
      .collect(Collectors.toList());
```

50. Lambda with Optional

```
Optional<String> optional = Optional.of("Java");
optional.ifPresent(System.out::println);  // Java
```

47. Lambda with Stream Limit

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka")
stream.limit(2)
      .forEach(System.out::println);  // Java, Spring
```

48. Lambda with Stream Skip

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka")
stream.skip(2)
      .forEach(System.out::println);  // Lambda, Kafka
```

49. Lambda with Stream Peek

```
Stream<String> stream = Stream.of("Java", "Spring", "Lambda", "Kafka")
stream.peek(System.out::println)
      .collect(Collectors.toList());
```

50. Lambda with Optional

```
Optional<String> optional = Optional.of("Java");
optional.ifPresent(System.out::println);  // Java
```