1. Which of the following is a correct syntax for a lambda expression?

a) `(a, b) -> a + b`

b) `(int a, int b) => a + b`

c) `(a, b) : a + b`

d) `a, b -> a + b`

---

2. A lambda expression can be assigned to:

a) An interface with only one abstract method

b) Any abstract class

c) Any interface

d) Only concrete classes

---

3. Identify the incorrect lambda expression :

a) `(x) -> x * 2`

b) `x -> { return x + 1; }`

c) `(x, y) -> { x + y }`

d) `(int x) -> x * x`

---

4. What is the return type of the following lambda?

```
(int x, int y) -> x + y
```

a) int

b) void

c) double

d) No return type

---

5. Lambda expressions can be used to instantiate:

a) Functional interfaces

b) Abstract classes

c) Enum types

d) Concrete classes

---

6. Choose the correct lambda for multiplying two numbers:

a) `(x, y) -> { x * y; }`

b) `(x, y) => x * y`

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

d) `x, y -> { return x * y }`

---

7. Which one is an invalid lambda syntax?

a) `() -> System.out.println("Hello" )`

b) `(String s) -> { System.out.println(s); }`

c) `(int x, int y) -> { return x * y }`

d) `x -> x + 1`

---

8. Lambda expressions can have how many abstract methods in the targettype?

a) One

b) Two
c) Three
d) Unlimited

---

9. Lambda expressions can capture:

a) Only instance variables
b) Only static variables
c) Final or effectively final variables
d) Any variable freely

---

10. Find the lambda that has a syntax error:
a) `(int x, int y) -> { return x + y; }`
b) `(int x, y) -> x + y`
c) `(x, y) -> { return x + y; }`
d) `(x) -> x * x`

---

11. What happens if you use a non-final local variable inside a lambda?a)

It compiles normally

b) Compilation error
c) Runtime error
d) Automatically becomes final

---

12. Which lambda correctly represents a method that accepts no parameters and returns a string?

a) `() -> "Hello"`
b) `-> "Hello"`
c) `( ) => "Hello"`
d) `() : "Hello"`

---

13. Choose the valid lambda expression :

a) `n -> n + 10`
b) `(n) -> { return n + 10 }`
c) `int n -> n + 10`
d) `n => n + 10`

---

14. Lambda expressions were introduced in which Java version? a)

Java 6

b) Java 7
c) Java 8
d) Java 9
15. Which of these is NOT true about lambda expressions?

a) They provide a clear and concise way to represent a method

---

b) They can have multiple abstract methods inside the interface

c) They can be used to implement functional interfaces
d) They can capture outer variables if they are effectively final

---

16. A lambda expression `(int a, int b) -> a + b` corresponds to which kind of method?

a) Takes two ints and returns an int
b) Takes two ints and returns void
c) Takes two Strings and returns a String
d) Takes no arguments

---

17. Select the incorrect way of writing a lambda with no parameters:

a) `() -> System.out.println("No parameters" )`
b) `() => System.out.println("No parameters" )`
c) `() -> { System.out.println("No parameters"); }`
d) `( ) -> "Done"`

---

18. Which functional interface matches a lambda that returns a booleanvalue?

a) Runnable
b) Predicate
c) Supplier
d) Consumer

---

19. Which lambda is incorrectly written?

a) `(a, b) -> a > b`
b) `(a, b) -> { return a > b; }`
c) `(a, b) : a > b`
d) `(a, b) -> (a > b)`

---

20. Which lambda expression is invalid?

a) `(int x) -> x + 1`
b) `(x, y) -> x - y`
c) `(int x, int y) -> { x + y; }`
d) `() -> { return 100; }`

Descriptive Scenario 1:

Task:
Write a lambda expression that accepts two integers and returns their sum.

Requirement:
Use the predefined functional interface `BiFunction<Integer, Integer, Integer>` to implement and test the lambda.

Descriptive Scenario 2:

Task: Create a lambda expression that takes no arguments and prints "Processing complete." **Requirement:**

Use the predefined functional interface `Supplier<String>`, and print the supplied value.

Descriptive Scenario 3:

Task:
Write a lambda expression that checks whether a given integer is even.

Requirement:
Use the predefined functional interface `Predicate<Integer>`. The lambda should return `true` if the number is even, otherwise `false`.

Descriptive Scenario 4:
Task:
Create a lambda expression that takes a `String` and returns its length.

Requirement:
Use the predefined functional interface `Function<String, Integer>` to implement and test this functionality.

Descriptive Scenario 5:

Task:
Develop a lambda expression that takes a floating-point number (`Float`) and prints whether it is positive or negative.

Requirement:
Use the predefined functional interface `Consumer<Float>`, and print an appropriate message like "Positive" or "Negative".

Task 1:

```
import java.util.function.BiFunction;
public class Task1 {
public static void main(String[] args) {
BiFunction<Integer,Integer,Integer> bifun=(a,b)->a+b;
System.out.println(bifun.apply(10, 20));
}}
```

Task 2:

```
import java.util.function.Supplier;
public class Task2 {
public static void main(String[] args) {
Supplier<String> sup=()->"Processing complete.";
System.out.println(sup.get());
}}
```

Task 3:

```
import java.util.function.Predicate;
public class Task3 {
public static void main(String[] args) {
Predicate<Integer> pre=(a)->a%2==0;
System.out.println(pre.test(10)?"Even":"Odd");
}}
```

Task 4:

```
import java.util.function.Function;
public class Task4 {
public static void main(String[] args) {
Function<String,Integer> fun=s->s.length();
System.out.println("The Lenght Of Given String Is "+fun.apply("Hello"));
}}
```

Task5:

```
import java.util.function.Consumer;
public class Task5 {
public static void main(String[] args) {
Consumer<Float> con=f-> System.out.println(f>0?"Positive Number":((f<0)?"Negative
Number":"Zero"));
con.accept(-10f);
}}
```