Task 1: Create an Anonymous Class for Vehicle

1. **Create an interface** named `Vehicle` with one method:
   - `void start();`
2. **In your `main()` method**:
   - Create an instance of `Vehicle` using an **anonymous class**.
   - Override the `start` method inside the anonymous class to print a message like:
     `"The vehicle is starting!"`
3. **Call the `start` method** to see the output.

Expected Output:

```
The vehicle is starting!
```

Task 2: Create an Anonymous Class for Calculator

1. **Create an interface** named `Calculator` with one method:
   - `int add(int a, int b);`
2. **In your `main()` method**:
   - Create an instance of `Calculator` using an **anonymous class**.
   - Override the `add` method inside the anonymous class to return the sum of two integers.
3. **Call the `add` method** with two numbers and print the result.

Expected Output:

```
The sum is: 15
```

Task 3: Create an Anonymous Class for Logging

1. **Create an interface** named `Logger` with one method:
   - `void log(String message);`
2. **In your `main()` method**:
   - Create an instance of `Logger` using an **anonymous class**.
   - Override the `log` method to print a custom log message like:
     `"Log: <message>"`
3. **Call the `log` method** and pass any log message to see the output.

Expected Output:

```
Log: System error occurred
```

Task 4: Create an Anonymous Class for Timer

1. **Create an interface** named `Timer` with one method:
    - `void startTimer(int seconds);`
2. **In your `main()` method**:
    - Create an instance of `Timer` using an **anonymous class**.
    - Override the `startTimer` method to simulate starting a timer and print: `"Timer started for <seconds> seconds."`
3. **Call the `startTimer` method** with a time value and see the output.

Expected Output:

```
Timer started for 30 seconds.
```

---

1. Which of the following is a valid functional interface?

- a) `interface A { void method(); void method2(); }`
- b) `interface A { void method(); }`
- c) `interface A { void method(); void method2(); void method3(); }`
- d) `interface A { void method1(); void method2(); }`

---

2. Which annotation is used to indicate that an interface is a functional interface?

- a) `@FunctionalInterface`
- b) `@Lambda`
- c) `@Interface`
- d) `@Functional`

---

3. A functional interface can have:

- a) Only one abstract method
- b) Only one default method
- c) Only one static method
- d) Any number of abstract methods

---

4. What will happen if a functional interface has more than one abstract method?

- a) It will still compile without issues
- b) It will throw a compile-time error
- c) It will throw a runtime exception
- d) It will be treated as a non-functional interface

---

5. Which of the following is an example of a built-in functional interface in Java?

- a) `Runnable`

- b) `Thread`
- c) `ActionListener`
- d) `Callable`

---

6. Can a functional interface have multiple default methods?

- a) No, it can only have one default method
- b) Yes, it can have multiple default methods
- c) Yes, but only if the default methods are private
- d) No, default methods are not allowed in functional interfaces

---

7. Can a functional interface extend another functional interface?

- a) Yes, it can extend multiple functional interfaces
- b) No, a functional interface cannot extend another functional interface
- c) Yes, but only one functional interface
- d) No, functional interfaces cannot extend any interfaces

---

8. Which of the following is true about functional interfaces in Java?

- a) They must contain exactly one abstract method
- b) They must contain at least one abstract method
- c) They must contain only default methods
- d) They must contain only static methods

---

9. What is the purpose of the `default` keyword in a functional interface?

- a) To define a method with a default implementation
- b) To define a method that is not overridden in the subclass
- c) To indicate that a method is abstract
- d) To make the method private
