```
@FunctionalInterface
public interface StringOperation {
    String operate(String s);
}

public class Main {
    public static void main(String[] args) {
        // Lambda expression to reverse a string
        StringOperation reverse = s -> new StringBuilder(s).reverse().toString();

        // Example usage
        String result = applyOperation("hello", reverse);
        System.out.println(result); // Outputs: "olleh"
    }

    // Method to apply the operation
    public static String applyOperation(String s, StringOperation operation) {
        return operation.operate(s);
    }
}
```

```
@FunctionalInterface
public interface ArithmeticOperation {
    double operate(double a, double b);
}

public class Main {
    public static void main(String[] args) {
        // Lambda expressions for arithmetic operations
        ArithmeticOperation addition = (a, b) -> a + b;
        ArithmeticOperation subtraction = (a, b) -> a - b;
        ArithmeticOperation multiplication = (a, b) -> a * b;
```

```
        ArithmeticOperation division = (a, b) -> a / b;

        // Testing the operations
        System.out.println("Addition: " + performOperation(10, 5, addition)); // 15.0
        System.out.println("Subtraction: " + performOperation(10, 5, subtraction)); // 5.0
        System.out.println("Multiplication: " + performOperation(10, 5, multiplication)); // 50.0
        System.out.println("Division: " + performOperation(10, 5, division)); // 2.0
    }

    // Method to perform the operation
    public static double performOperation(double a, double b, ArithmeticOperation operation) {
        return operation.operate(a, b);
    }
}
```

```
@FunctionalInterface
public interface StringTransform {
    String transform(String s);
}

public class Main {
    public static void main(String[] args) {
        // Lambda expression to convert a string to uppercase
        StringTransform toUpperCase = String::toUpperCase;

        // Lambda expression to reverse a string
        StringTransform reverse = s -> new StringBuilder(s).reverse().toString();

        // Testing the lambdas
        String testString = "Hello World";
        System.out.println("Uppercase: " + toUpperCase.transform(testString)); // "HELLO
WORLD"
        System.out.println("Reversed: " + reverse.transform(testString)); // "dlroW olleH"
```

```
    }
}

@FunctionalInterface
public interface StringTest {
    boolean test(String s);
}

public class Main {
    public static void main(String[] args) {
        // Lambda to check if a string is a palindrome
        StringTest isPalindrome = s -> s.equals(new StringBuilder(s).reverse().toString());

        // Lambda to check if a string contains a specific character ('a' in this case)
        StringTest containsCharacter = s -> s.contains("a");

        // Testing the lambdas
        String palindrome = "radar";
        String nonPalindrome = "hello";
        System.out.println("Is 'radar' a palindrome? " + isPalindrome.test(palindrome)); // true
        System.out.println("Is 'hello' a palindrome? " + isPalindrome.test(nonPalindrome)); // false
        System.out.println("Does 'hello' contain 'a'? " + containsCharacter.test(nonPalindrome)); // false
    }
}
```

```
@FunctionalInterface
public interface Logger {
    void log(String message);
}
```

```
public class Main {
    public static void main(String[] args) {
        // Lambda expressions for different logging levels
        Logger infoLogger = message -> System.out.println("INFO: " + message);
        Logger debugLogger = message -> System.out.println("DEBUG: " + message);
        Logger errorLogger = message -> System.out.println("ERROR: " + message);

        // Testing the loggers
        logMessage("This is an informational message.", infoLogger);
        logMessage("This is a debug message.", debugLogger);
        logMessage("This is an error message.", errorLogger);
    }

    // Method to log messages
    public static void logMessage(String message, Logger logger) {
        logger.log(message);
    }
}
```