Question 1

This task focuses on combining 2 arrays together by multiplying the elements of the shorter array with the corresponding elements in the longer array and then filling the remaining positions with values from the longer array values. The code begins by determining the length of the shorter array and then identifies the longer array. Using 2 for loops, it iterates through the shorter array to compute the products for overlapping elements and then populates the rest of the output array with the remaining values from the longer array. The JUnit test cases verify the correctness of the mult() method by using assertEquals to compare the expected and actual output arrays.

Question 2

This task already contained an error in the given code.To resolve this, I looked up Heron's formula online and found out that the s, and side variables are flipped. The JUnit test class includes the following methods, listed in order: testing the validity of a Triangle with sides (3, 4, 100), testing the calculation for the areas of t1, t2, and t3, verifying if t1 and t2 are equal, and a test to show what happens if the length is negative, ensuring it cannot be misused.

Question 3

The program checks whether a given string is a valid phone number using a regular expression (regex). The regex defines a specific format the phone number must follow. The task involves fixing and testing the regex to handle various valid and invalid phone number formats.

Final Regex

\\(\\d{3}\\)\\s*\\d{3}\\s*-\\s*\\d{4}

\\(: Matches the opening parenthesis (. Parentheses are special characters in regex (used for capturing groups), so we need to escape them with \\.

\\d{3} : Matches exactly 3 digits (\d is shorthand for any digit [0-9]). {3} specifies that there must be exactly 3 occurrences of the preceding pattern (\d).

\\) : Matches the closing parenthesis). Like the opening parenthesis, it is escaped with \\.

\\s* : Matches zero or more whitespace characters (\s is shorthand for any whitespace, including spaces, tabs, and line breaks). The * quantifier allows for zero or more occurrences of the preceding whitespace.

\\d{3} : Matches another group of exactly 3 digits.

\\s*-\\s* : Matches a hyphen (-) surrounded by zero or more whitespace characters on either side. The \\s* before and after the - allows for flexibility in spacing.

\\d{4}: Matches exactly 4 digits.

Question 4

This task introduces JUnit suites, which are simply empty classes that serve as containers for JUnit test classes. They help maintain clean and organized code by providing a centralized location where all tests for a specific project can be executed with a single click.

Question 5

This task focuses on parameterized testing with a constructor, basically a test class containing a test method and another method that provides preset parameters for testing. The @Parameterized.Parameters method provides test data in the form of a Collection<Object[]>. The constructor takes two parameters: input (the Fibonacci index) and expected (the expected Fibonacci value). The testFibonacci method uses assertEquals to compare the result of Fibonacci.compute(input) with the expected value. The data() method defines the input-output pairs for the first 10 Fibonacci numbers.

Question 6

This task is the same as Question 5 but testing for prime numbers. The only difference is the parameters. The parameters in this question are the numbers given from the lab manual which are integers and boolean values to indicate true if values are prime and false if they are not.

Question 7

This task introduces JUnit Theories, a concept similar to parameterized testing but with more limitations. Theories in JUnit are similar to mathematical theories, where a statement is proposed to be true for specific values and is tested against other values to validate its correctness. At the start of the question, the theory a + b > a and a + b > b is introduced. This theory is implemented in the code, and a set of values is used to test its validity. The theory includes an assertTrue test, which verifies that the condition inside the assertion is true. If the condition is false, the test will fail and execution will stop.

Question 8

This task converts the parameterized testing in Questions 5 and 6 into theory JUnit tests. Instead of setting parameters for a test, a theory is created so that the method's result matches the expected value from a given set of data. A set of two-element arrays is provided, where the first element is the input value and the second is the expected output. This is similar to the code in the parameterized testing but instead uses theories.