Static methods Lab

Task 1: Explore the parts of a Method

Begin by taking a look at the code that you've downloaded. **Discuss with your partner differences you notice in the structure of this program from the programs you've written so far.** Run this program a few times. Does it do what you expect it to? Recall that a prime number is an integer that is evenly divisible only by 1 and itself. The first few primes are 2, 3, 5, 7, and 11.

Now look at the method isPrime. How would you know that it is a static method? Methods are designed to work like mathematical functions such as "square root". For square root to work we need to provide information, a number, and we expect to get an answer back, the square root of the number we provided. Note that the isPrime method is given a number and returns an answer that says whether or not the given number is prime.

The isPrime method is used by the main method in the line of code copied below:

if (PrimeMethods.isPrime(userNumber))

What's underlined above is a **method call**. Static methods are called using the following form:

<ClassName>.<methodName>(<arguments>)

The information we provide when we call a method is referred to as the **parameter values** (also known as arguments). For our program, we provide the isPrime method the value in variable userNumber, which is underlined in the code below:

if (PrimeMethods.isPrime(userNumber))

Where does the isPrime method store this value? In its **parameter variable** (also known simply as a parameter). Methods store their parameter values in their parameter variables, which are listed in the method's header in the parentheses after the method name. Like variable declarations, parameter variables must first specify their type and then their name. In our code, the isPrime method has one parameter variable, named number, of type int, as underlined in the code:

public static boolean isPrime(**int** number)

A method can have more than one parameter if needed, or zero parameters if no information needs to be provided to the method.

We also need to specify what type of answer is returned by a method. We call this the **return type**, and it is listed in the method header right before the method's name. In our code, the isPrime method specifies it will return a boolean answer, as underlined below:

public static boolean isPrime(**int** number)

If a method doesn't return anything, we use the type void. Notice the return type of the main method.

The method's body, enclosed in braces, follows after the method's header. Here's where we put the statements that implement the method's algorithm so that it does the task indicated by the method's name. Take a look at the code that determines if a number is prime. Our implementation is very simple. (One of the challenges below is to improve efficiency of this method.) Together, a method header and its body are called a **method definition**.

Quiz your lab partner on the terms used to refer to each part of the PrimeFactor method.

Task 2: Use a Method

Let's give this a try! Beneath isPrime is another completed method called findMinPrimeFactor. This method has one parameter, just like isPrime, but it returns an int instead of a boolean.

Some Math: Every positive integer can be written as a product of prime numbers. This is a number's **prime factorization**. Each integer has a unique set of primes that produce it when multiplied together. Here are some examples of prime factorizations:

Integer	**Factorization**
4	2 * 2 (or 2^2)
11	11
15	3 * 5
24	2 * 2 * 2 * 3 (or 2^3 * 3)
39	3 * 13

Because a factorization is a product of primes, multiplying two numbers just combines their factors. So 60 = 4 * 15 = (2 * 2) * (3 * 5), and the prime factorization of 60 is 2 * 2 * 3 * 5.

First, add code to main that calls the findMinPrimeFactor method, using userNumber. Display the user's number, the number that the method returns and the value of the user's number divided by the return value. Next, modify your code to print the prime factorization of userNumber. To do this, you'll need to repeatedly find and display the minimum prime factor and divide the number by that factor until the number has been reduced to 1. Don't worry about formatting; just print all the factors the correct number of times.