Humphrey & Keith:

SOLUTION:

```
java
import java.util.Random;

public class FloatingPointConversion {
    public static void main(String[] args) {
        Random random = new Random();

        System.out.println("S/No.\tDecimal Number\t\tBinary Number\t\tRemarks\n");

        // Generate and process 50 random floating-point numbers
        for (int i = 1; i <= 50; i++) {
            // Generate a random decimal number between 0 and 1000 with at most
three decimal points
            double decimalNumber = random.nextDouble() * 1000.0;
            decimalNumber = Math.round(decimalNumber * 1000.0) / 1000.0; // Round to
three decimal points

            // Convert the decimal number to binary notation
            String binaryNumber = convertToBinary(decimalNumber);

            // Determine the remarks based on the binary number's length
            String remarks = getRemarks(binaryNumber);

            // Print the result in a formatted manner
            System.out.printf("%d.\t%.3f\t\t%s\t%s\n", i, decimalNumber,
binaryNumber, remarks);
        }
    }

    // Convert a decimal number to binary notation
    public static String convertToBinary(double decimal) {
        long intPart = (long) decimal;
        double fracPart = decimal - intPart;

        // Convert the integer part to binary
        StringBuilder binaryIntPart = new
StringBuilder(Long.toBinaryString(intPart));

        // Convert the fractional part to binary (up to 12 bits)
        StringBuilder binaryFracPart = new StringBuilder();
        for (int i = 0; i < 12; i++) {
            fracPart *= 2;
            binaryFracPart.append((int) fracPart);
            fracPart -= (int) fracPart;
        }

        return binaryIntPart + "." + binaryFracPart.toString();
    }

    // Determine the remarks (Approximate or Exactly) based on binary number's
length
    public static String getRemarks(String binaryNumber) {
        if (binaryNumber.length() >= 5) {
            return "Exactly";
        } else {
            return "Approximate";
        }
    }
}
```

Here's a breakdown of the key sections and functions in the code:

1. **Imports and Class Declaration**: Import the required libraries and define the `FloatingPointConversion` class.
2. **Main Method**: The program starts executing from the `main` method. A `Random` object is created to generate random numbers.
3. **Output Header**: A header is printed to label the columns of the output table.
4. **Loop for Random Numbers**: The program generates and processes 50 random floating-point numbers within a loop.
5. **Generating Random Decimal Number**: Inside the loop, a random decimal number is generated between 0 and 1000 using `random.nextDouble()` and then rounded to three decimal points.
6. **Converting to Binary**: The `convertToBinary` function converts a decimal number to binary notation. It separates the integer and fractional parts and converts them individually to binary strings.
7. **Converting Integer Part to Binary**: The integer part is converted to binary using `Long.toBinaryString(intPart).`
8. **Converting Fractional Part to Binary**: The fractional part is converted to binary by repeatedly multiplying it by 2 and appending the integer part of the result to the binary representation.
9. **Determining Remarks**: The `getRemarks` function determines whether the binary number has at least 5 digits (nibbles), classifying it as "Exactly" or "Approximate."
10. **Printing Results**: The results, including the serial number, decimal number, binary number, and remarks, are printed in a formatted manner using `System.out.printf`.
11. **Convert to Binary Function**: The `convertToBinary` function handles the conversion of a decimal number to binary notation. It returns the combined binary representation of the integer and fractional parts.
12. **Get Remarks Function**: The `getRemarks` function determines the remarks for a given binary number based on its length.

This program uses functions to encapsulate the logic for different tasks, making the code modular and easier to understand. It generates random numbers, converts them to binary, and provides remarks based on the binary representation's length, similar to the requirements in "Table Q1 (b)."