Question 1: Tracing Integer and Floating-Point Arithmetic with Casting

What is the final output of the following Java code segment?

```
public class ArithmeticTrace {
   public static void main(String[] args) {
      int a = 17;
      int b = 5;
      double result1 = (double) a / b;
      int result2 = a / b;
      int remainder = a % b;
      double castedRemainder = (double) (a % b);
      int finalResult = (int) (result1 + remainder);
      System.out.println("Result 1 (double division): " + result1);
      System.out.println("Result 2 (integer division): " + result2);
      System.out.println("Remainder: " + remainder);
      System.out.println("Casted Remainder (double): " + castedRemainder);
      System.out.println("Final Result (int after operations): " + finalResult);
   }
}
```

This question draws on concepts from Chapter 2, focusing on elementary programming basics such as integer and floating-point division, the remainder operator, and explicit type casting.

Question 2: Implementing Conditional Logic with Grade Ranges

Write a complete Java program named LetterToNumericGrade that prompts the user to enter a numeric grade (an integer between 0 and 100, inclusive). The program should then use if-else if statements to display the corresponding letter grade (A, B, C, D, or F) based on the following scale:

- 90-100: A
- 80-89: B
- 70-79: C
- 60-69: D
- 0-59: F

If the entered numeric grade is outside the 0-100 range, the program should print "Invalid Grade" and not assign a letter grade.

This question tests your understanding of selection statements, specifically if-else if structures, as discussed in Chapter 3.

Question 3: Tracing Loops with continue and Random Numbers

Consider the following Java code. Assume Math.random() generates numbers such that (int)(Math.random() * 5) + 1 produces values from 1 to 5. What will be printed to the console when the main method is executed?

import java.util.Random; // Assume this import is present for demonstration, though Math.random() is used

```
public class LoopAndRandomTrace {
   public static void main(String[] args) {
      int sum = 0;
      for (int i = 1; i <= 5; i++) {
         int randomNumber = (int)(Math.random() * 5) + 1; // Generates a random integer from 1 to 5
         System.out.println("Iteration " + i + ": Random Number = " + randomNumber);

         if (randomNumber % 2 != 0) { // If number is odd
            System.out.println("  Skipping odd number.");
            continue;
```

```
        }
        if (randomNumber == 4) {
            System.out.println("  Encountered 4, stopping loop.");
            break;
        }
        sum += randomNumber;
        System.out.println("  Current Sum = " + sum);
    }
    System.out.println("Final Sum: " + sum);
  }
}
```

This question requires tracing a for loop, understanding the behavior of continue and break statements, and how Math.random() generates values, as covered in Chapters 4 and 5. Note that for tracing, you would typically be given a fixed sequence of random numbers, but for this make-up question, the behavior of Math.random() producing values from 1 to 5 is described.

Question 4: Understanding Method Parameters and Variable Scope

Consider the following Java code. What will be printed to the console when the main method is executed?

```
public class MethodScopeExample {


  public static void main(String[] args) {
    int valueA = 100;
    String message = "Hello";
    System.out.println("Before call: valueA = " + valueA + ", message = " + message);
    // Call the modifyData method
```

```
        modifyData(valueA, message);

        System.out.println("After call: valueA = " + valueA + ", message = " + message);

        { // Start of an inner block
            int valueB = 50;
            System.out.println("Inside inner block: valueA = " + valueA + ", valueB = " + valueB);
        } // End of inner block
        // System.out.println("Outside inner block: valueB = " + valueB); // This line would cause a compile-time error if uncommented
    }
    public static void modifyData(int num, String text) {
        num = num + 20;
        text = text + " World";
        System.out.println("Inside method: num = " + num + ", text = " + text);
    }
}
```

This question tests your knowledge of method calls, pass-by-value semantics for primitive types and object references, and the scope of local variables within methods and nested blocks, as detailed in Chapter 6.

Question 5: Array Traversal, Summation, and Finding Extremes

Consider the following Java code. What will be printed to the console?

```
public class ArrayAnalyzer {
    public static void main(String[] args) {
        int[] numbers = {15, 7, 22, 10, 5, 18, 3};
```

```
        int sum = 0;
        int max = numbers;
        int min = numbers;
        for (int i = 0; i < numbers.length; i++) {
            sum += numbers[i];
            if (numbers[i] > max) {
                max = numbers[i];
            }
            if (numbers[i] < min) {
                min = numbers[i];
            }
        }
        double average = (double) sum / numbers.length;
        System.out.println("Sum of elements: " + sum);
        System.out.println("Maximum element: " + max);
        System.out.println("Minimum element: " + min);
        System.out.println("Average of elements: " + average);
    }
}
```

This question assesses your ability to declare and initialize arrays, use for loops to iterate through array elements, perform basic arithmetic operations (summation, division for average), and find maximum and minimum values, concepts detailed in Chapters 5 and 7.

Question 6: String and Character Analysis with Conditionals

What is the output of the following Java program when executed?

```
public class CharacterProcessor {
    public static void main(String[] args) {
```

```
        String phrase = "Java Programming 2023!";
        int uppercaseCount = 0;
        int digitCount = 0;
        StringBuilder modifiedPhrase = new StringBuilder();

        for (int i = 0; i < phrase.length(); i++) {
            char ch = phrase.charAt(i);

            if (Character.isUpperCase(ch)) {
                uppercaseCount++;
                modifiedPhrase.append(Character.toLowerCase(ch));
            } else if (Character.isDigit(ch)) {
                digitCount++;
                modifiedPhrase.append('*');
            } else {
                modifiedPhrase.append(ch);
            }
        }

        System.out.println("Original Phrase: " + phrase);
        System.out.println("Uppercase Letters Found: " + uppercaseCount);
        System.out.println("Digits Found: " + digitCount);
        System.out.println("Modified Phrase: " + modifiedPhrase.toString());
    }
}
```

This question tests your understanding of String manipulation, character-by-character processing using for loops, and the application of Character class methods like isUpperCase, isDigit, and toLowerCase within if-else if constructs, as discussed in Chapters 4 and 3.

Question 7: Nested Loops for Pattern Generation

Trace the execution of the following Java code segment and determine its exact output.

```
public class NestedPattern {
  public static void main(String[] args) {
    int rows = 4;
    for (int i = 1; i <= rows; i++) {
      for (int j = 1; j <= i; j++) {
        System.out.print(j + " ");
      }
      System.out.println();
    }
  }
}
```

This question focuses on your ability to trace nested for loops and predict their output, particularly when generating patterns where the inner loop's iterations depend on the outer loop's current iteration, a concept explained in Chapter 5 on nested loops.

Question 8: Simple Class Definition and Object Interaction

Consider the two Java classes defined below. What is the complete output when the MainApp class's main method is executed?

```
// File: MyGreeter.java
public class MyGreeter {
  String greeting;
```

```
    public MyGreeter(String text) {
        this.greeting = text;
    }

    public void displayGreeting(String name) {
        System.out.println(greeting + ", " + name + "!");
    }
}

// File: MainApp.java
public class MainApp {
    public static void main(String[] args) {
        MyGreeter morningGreeter = new MyGreeter("Good Morning");
        MyGreeter eveningGreeter = new MyGreeter("Good Evening");

        morningGreeter.displayGreeting("Alice");
        eveningGreeter.displayGreeting("Bob");
    }
}
```

This question tests your understanding of fundamental object-oriented programming concepts, including defining a class with a constructor and a method, instantiating objects from that class, and invoking methods on those objects from the main method, as covered in Chapter 9 on Objects and Classes and Chapter 6 on Methods.

Answers to All 8 Questions

Here are the answers to the questions, with detailed explanations and source citations:

Answer to Question 1: Tracing Integer and Floating-Point Arithmetic with Casting

This question focuses on how Java handles arithmetic operations, especially integer division, the remainder operator, and type casting.

- int a = 17;
- int b = 5;

1. double result1 = (double) a / b;
    - Here, a (an int) is explicitly cast to a double (17.0).
    - Then, 17.0 (double) is divided by b (5, which is promoted to 5.0 for the double division).
    - Result: 17.0 / 5.0 yields 3.4.
2. int result2 = a / b;
    - This is integer division because both a and b are int types.
    - 17 / 5 yields 3 (the fractional part is truncated).
3. int remainder = a % b;
    - The remainder operator (%) returns the remainder of the division.
    - 17 % 5 means 17 = 3 * 5 + 2, so the remainder is 2.
4. double castedRemainder = (double) (a % b);
    - First, a % b is evaluated, which is 2.
    - Then, 2 is cast to a double, resulting in 2.0.
5. int finalResult = (int) (result1 + remainder);
    - result1 is 3.4.
    - remainder is 2.
    - 3.4 + 2 yields 5.4.
    - This sum (5.4) is then explicitly cast to an int. When casting a double to an int, the fractional part is truncated.

- Result: 5.

Output:

Result 1 (double division): 3.4

Result 2 (integer division): 3

Remainder: 2

Casted Remainder (double): 2.0

Final Result (int after operations): 5

This example illustrates integer division and remainder operator behavior, as well as the effects of explicit type casting between int and double.

Answer to Question 2: Implementing Conditional Logic with Grade Ranges

This question requires the use of if-else if statements to implement selection logic based on grade ranges. The program also needs to handle invalid input.

Complete Java Program (LetterToNumericGrade.java):

```
import java.util.Scanner;


public class LetterToNumericGrade {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);


    System.out.print("Enter a numeric grade (0-100): ");
    int numericGrade = input.nextInt();


    String letterGrade;


    if (numericGrade < 0 || numericGrade > 100) {
```

```
        letterGrade = "Invalid Grade";
    } else if (numericGrade >= 90) {
        letterGrade = "A";
    } else if (numericGrade >= 80) {
        letterGrade = "B";
    } else if (numericGrade >= 70) {
        letterGrade = "C";
    } else if (numericGrade >= 60) {
        letterGrade = "D";
    } else { // numericGrade is between 0 and 59
        letterGrade = "F";
    }

    System.out.println("Corresponding Letter Grade: " + letterGrade);

    input.close();
  }
}
```

Explanation: The program first prompts the user for a numeric grade and reads it using a Scanner object. It then uses a series of if-else if statements to determine the letter grade. The conditions are ordered from highest grade range to lowest, or from invalid range outwards, to ensure correct evaluation.

1. The first if condition (numericGrade < 0 || numericGrade > 100) checks for invalid input. If the grade is outside the 0-100 range, it prints "Invalid Grade". Logical OR (||) means the condition is true if *either* sub-condition is true.
2. If the input is valid, the program proceeds to check the specific grade ranges using else if. For example, numericGrade >= 90 checks for an 'A'. Because of the else if

structure, this condition is only evaluated if numericGrade is *not* less than 0 or greater than 100.

3. The final else block catches any valid grade that falls below 60, assigning an 'F'. This structure ensures that only one block of code is executed based on the first true condition encountered.

Answer to Question 3: Tracing Loops with continue and Random Numbers

This question involves tracing a for loop, understanding continue and break statements, and how Math.random() works in this context. Assume (int)(Math.random() * 5) + 1 generates a sequence of random integers from 1 to 5. Since the output depends on the random numbers, let's assume a specific sequence for a clear trace.

Assumed Random Number Sequence for Tracing: Let's say randomNumber generates the following sequence for i = 1 to 5: 3, 4, 1, 2, 5 (this is an arbitrary sequence chosen for demonstration, as Math.random() produces unpredictable results).

Trace:

- sum = 0;
- **Iteration 1 (i = 1):**
    - randomNumber is 3.
    - System.out.println("Iteration 1: Random Number = 3");
    - if (randomNumber % 2 != 0) (i.e., 3 % 2 != 0 which is 1 != 0, True).
    - System.out.println(" Skipping odd number.");
    - continue; (skips the rest of the current iteration and goes to the next i).
    - sum remains 0.
- **Iteration 2 (i = 2):**
    - randomNumber is 4.
    - System.out.println("Iteration 2: Random Number = 4");
    - if (randomNumber % 2 != 0) (i.e., 4 % 2 != 0 which is 0 != 0, False).

- if (randomNumber == 4) (i.e., 4 == 4, True).
- System.out.println(" Encountered 4, stopping loop.");
- break; (immediately terminates the loop).
- The loop ends.

- **After loop:**
  - System.out.println("Final Sum: " + sum);

Output (based on assumed random sequence 3, 4):

Iteration 1: Random Number = 3

Skipping odd number.

Iteration 2: Random Number = 4

Encountered 4, stopping loop.

Final Sum: 0

This demonstrates the continue keyword skipping the current iteration's remaining code block and moving to the next loop iteration, and the break keyword terminating the entire loop prematurely. Without break, the loop would continue to i=5. Math.random() generates a double value greater than or equal to 0.0 and less than 1.0. Multiplying by 5 and casting to int gives values from 0 to 4, so adding 1 shifts the range to 1 to 5.

Answer to Question 4: Understanding Method Parameters and Variable Scope

This question explores Java's pass-by-value mechanism for method parameters and the concept of variable scope.

Trace:

1. main method starts.
2. int valueA = 100; [100 is assigned to valueA]
3. String message = "Hello"; ["Hello" is assigned to message]
4. System.out.println("Before call: valueA = " + valueA + ", message = " + message);

- Output: Before call: valueA = 100, message = Hello

5. modifyData(valueA, message);
    - The modifyData method is called.
    - valueA (100) is passed by value to num. num becomes 100.
    - message ("Hello") is passed by value to text. text becomes a copy of the *reference* to the "Hello" string. Java passes object references by value, meaning the text parameter *points to the same string object* as message initially.
    - **Inside modifyData:**
        - num = num + 20; (num becomes 100 + 20 = 120). This change only affects the local copy num, not valueA in main.
        - text = text + " World"; This statement creates a *new* String object "Hello World" and text is reassigned to refer to this *new* object. The message variable in main still refers to the *original* "Hello" string object.
        - System.out.println("Inside method: num = " + num + ", text = " + text);
            - Output: Inside method: num = 120, text = Hello World
    - modifyData method finishes, control returns to main.
6. System.out.println("After call: valueA = " + valueA + ", message = " + message);
    - valueA remains 100 (its value was not changed by the method call).
    - message remains "Hello" (it still points to the original string, as the text parameter inside modifyData was reassigned to a *new* string object).
    - Output: After call: valueA = 100, message = Hello
7. { int valueB = 50; ... } (inner block)
    - int valueB = 50; (valueB is declared and initialized within this block). Its scope is limited to this block.
    - System.out.println("Inside inner block: valueA = " + valueA + ", valueB = " + valueB);
        - Output: Inside inner block: valueA = 100, valueB = 50

8. // System.out.println("Outside inner block: valueB = " + valueB);
    - If uncommented, this line would cause a compile-time error because valueB is out of scope here; it was declared within the inner block.

Complete Output:

Before call: valueA = 100, message = Hello

Inside method: num = 120, text = Hello World

After call: valueA = 100, message = Hello

Inside inner block: valueA = 100, valueB = 50

This illustrates that Java passes primitive types by value (a copy of the actual value), and object references by value (a copy of the reference). Changes to the *value* of a primitive parameter or the *reference* of an object parameter within a method do not affect the original variable outside that method. Variable scope determines where a variable can be accessed.

Answer to Question 5: Array Traversal, Summation, and Finding Extremes

This question demonstrates basic array processing using a for loop to calculate sum, find maximum, minimum, and average [7.3, 244, 245].

Trace:

- int[] numbers = {15, 7, 22, 10, 5, 18, 3}; (Array numbers is initialized)
- int sum = 0;
- int max = numbers; (max is 15)
- int min = numbers; (min is 15)

Loop (for (int i = 0; i < numbers.length; i++)): numbers.length is 7.

- **i = 0:**
    - numbers is 15.
    - sum += 15; (sum becomes 15)
    - 15 > max (15 > 15) is False.

    - 15 < min (15 < 15) is False.
- **i = 1:**
    - numbers is 7.
    - sum += 7; (sum becomes 15 + 7 = 22)
    - 7 > max (7 > 15) is False.
    - 7 < min (7 < 15) is True. min becomes 7.
- **i = 2:**
    - numbers is 22.
    - sum += 22; (sum becomes 22 + 22 = 44)
    - 22 > max (22 > 15) is True. max becomes 22.
    - 22 < min (22 < 7) is False.
- **i = 3:**
    - numbers is 10.
    - sum += 10; (sum becomes 44 + 10 = 54)
    - 10 > max (10 > 22) is False.
    - 10 < min (10 < 7) is False.
- **i = 4:**
    - numbers is 5.
    - sum += 5; (sum becomes 54 + 5 = 59)
    - 5 > max (5 > 22) is False.
    - 5 < min (5 < 7) is True. min becomes 5.
- **i = 5:**
    - numbers is 18.
    - sum += 18; (sum becomes 59 + 18 = 77)
    - 18 > max (18 > 22) is False.
    - 18 < min (18 < 5) is False.

- **i = 6:**
    - numbers is 3.
    - sum += 3; (sum becomes 77 + 3 = 80)
    - 3 > max (3 > 22) is False.
    - 3 < min (3 < 5) is True. min becomes 3.

After loop:

- double average = (double) sum / numbers.length;
    - average = (double) 80 / 7;
    - average = 11.428571428571429 (approximately)

Output:

Sum of elements: 80

Maximum element: 22

Minimum element: 3

Average of elements: 11.428571428571429

This program demonstrates iterating through an array to perform calculations. The array elements are accessed using an index within the loop, from 0 to numbers.length - 1 [7.2.6, 243]. Summation involves sum += numbers[i]. Finding max/min involves simple conditional checks. Calculating the average requires casting sum to a double to ensure floating-point division.

Answer to Question 6: String and Character Analysis with Conditionals

This question tests character processing using String.charAt() and methods from the Character class, combined with if-else if statements.

Trace:

- String phrase = "Java Programming 2023!";
- int uppercaseCount = 0;
- int digitCount = 0;
- StringBuilder modifiedPhrase = new StringBuilder();

Loop (for (int i = 0; i < phrase.length(); i++)): phrase.length() is 21.

- **i = 0, ch = 'J':** Character.isUpperCase('J') is True.
    - uppercaseCount becomes 1.
    - modifiedPhrase.append(Character.toLowerCase('J')) appends 'j'. (modifiedPhrase: "j")
- **i = 1, ch = 'a':** Character.isUpperCase('a') is False. Character.isDigit('a') is False.
    - modifiedPhrase.append('a'). (modifiedPhrase: "ja")
- **i = 2, ch = 'v':** Character.isUpperCase('v') is False. Character.isDigit('v') is False.
    - modifiedPhrase.append('v'). (modifiedPhrase: "jav")
- **i = 3, ch = 'a':** Character.isUpperCase('a') is False. Character.isDigit('a') is False.
    - modifiedPhrase.append('a'). (modifiedPhrase: "java")
- **i = 4, ch = ' ':** Character.isUpperCase(' ') is False. Character.isDigit(' ') is False.
    - modifiedPhrase.append(' '). (modifiedPhrase: "java ")
- **i = 5, ch = 'P':** Character.isUpperCase('P') is True.
    - uppercaseCount becomes 2.
    - modifiedPhrase.append(Character.toLowerCase('P')) appends 'p'. (modifiedPhrase: "java p")
- **i = 6, ch = 'r':** Character.isUpperCase('r') is False. Character.isDigit('r') is False.
    - modifiedPhrase.append('r'). (modifiedPhrase: "java pr")
- ... (similar for 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g')
    - uppercaseCount increments for 'P'.
    - modifiedPhrase accumulates "java programming" (all lowercase).
- **i = 16, ch = ' ':** modifiedPhrase.append(' '). (modifiedPhrase: "java programming ")
- **i = 17, ch = '2':** Character.isUpperCase('2') is False. Character.isDigit('2') is True.
    - digitCount becomes 1.
    - modifiedPhrase.append('*'). (modifiedPhrase: "java programming *")
- **i = 18, ch = '0':** Character.isUpperCase('0') is False. Character.isDigit('0') is True.

    - digitCount becomes 2.
    - modifiedPhrase.append('*'). (modifiedPhrase: "java programming **")
- **i = 19, ch = '2':** Character.isUpperCase('2') is False. Character.isDigit('2') is True.
    - digitCount becomes 3.
    - modifiedPhrase.append('*'). (modifiedPhrase: "java programming ***")
- **i = 20, ch = '3':** Character.isUpperCase('3') is False. Character.isDigit('3') is True.
    - digitCount becomes 4.
    - modifiedPhrase.append('*'). (modifiedPhrase: "java programming ****")
- **i = 21, ch = '!':** Character.isUpperCase('!') is False. Character.isDigit('!') is False.
    - modifiedPhrase.append('!'). (modifiedPhrase: "java programming ****!")

Final Values:

- uppercaseCount: 2 (for 'J' and 'P')
- digitCount: 4 (for '2', '0', '2', '3')
- modifiedPhrase: "java programming ****!"

Output:

Original Phrase: Java Programming 2023!

Uppercase Letters Found: 2

Digits Found: 4

Modified Phrase: java programming ****!

This example shows how to iterate through a String using charAt(index). It utilizes Character class methods like isUpperCase() and isDigit() for checking character properties, and toLowerCase() for conversion. StringBuilder is used for efficient modification of the string. The if-else if-else structure processes characters based on defined conditions.

Answer to Question 7: Nested Loops for Pattern Generation

This question is about tracing nested for loops, where the inner loop's behavior is dependent on the outer loop's variable, to produce a specific pattern [5.9, 200, 49].

Trace:

- int rows = 4;

Outer loop (for (int i = 1; i <= rows; i++)):

- **i = 1:**
    - Inner loop (for (int j = 1; j <= i; j++)): j goes from 1 to 1.
        - j = 1: System.out.print(1 + " "); (prints "1 ")
    - System.out.println(); (moves to next line)
- **i = 2:**
    - Inner loop (for (int j = 1; j <= i; j++)): j goes from 1 to 2.
        - j = 1: System.out.print(1 + " "); (prints "1 ")
        - j = 2: System.out.print(2 + " "); (prints "2 ")
    - System.out.println(); (moves to next line)
- **i = 3:**
    - Inner loop (for (int j = 1; j <= i; j++)): j goes from 1 to 3.
        - j = 1: System.out.print(1 + " "); (prints "1 ")
        - j = 2: System.out.print(2 + " "); (prints "2 ")
        - j = 3: System.out.print(3 + " "); (prints "3 ")
    - System.out.println(); (moves to next line)
- **i = 4:**
    - Inner loop (for (int j = 1; j <= i; j++)): j goes from 1 to 4.
        - j = 1: System.out.print(1 + " "); (prints "1 ")
        - j = 2: System.out.print(2 + " "); (prints "2 ")
        - j = 3: System.out.print(3 + " "); (prints "3 ")
        - j = 4: System.out.print(4 + " "); (prints "4 ")

- System.out.println(); (moves to next line)

Output:

1

1 2

1 2 3

1 2 3 4

This pattern, often called a right-aligned triangle or pyramid, is a classic example of using nested loops. The outer loop controls the number of rows (or lines of output), and the inner loop controls what is printed on each row. In this case, the inner loop's condition j <= i ensures that the number of elements printed on each line is equal to the current row number (i). Each System.out.println() after the inner loop ensures that each row starts on a new line.

Answer to Question 8: Simple Class Definition and Object Interaction

This question assesses the fundamental concepts of Object-Oriented Programming (OOP) in Java: defining a class, creating a constructor, defining a method, instantiating objects, and calling methods on those objects.

Trace:

1. MainApp class's main method starts execution.
2. MyGreeter morningGreeter = new MyGreeter("Good Morning");
    - A new MyGreeter object is created.
    - The constructor MyGreeter(String text) is called with "Good Morning".
    - Inside the constructor, this.greeting (the greeting field of the morningGreeter object) is set to "Good Morning".
    - morningGreeter now refers to this new object.
3. MyGreeter eveningGreeter = new MyGreeter("Good Evening");
    - Another new MyGreeter object is created.

- The constructor MyGreeter(String text) is called with "Good Evening".
- Inside the constructor, this.greeting (the greeting field of the eveningGreeter object) is set to "Good Evening".
- eveningGreeter now refers to this second new object.

4. morningGreeter.displayGreeting("Alice");
    - The displayGreeting method is called on the morningGreeter object.
    - The name parameter is "Alice".
    - Inside the displayGreeting method, System.out.println(greeting + ", " + name + "!"); is executed.
    - greeting for morningGreeter is "Good Morning".
    - Output: Good Morning, Alice!
5. eveningGreeter.displayGreeting("Bob");
    - The displayGreeting method is called on the eveningGreeter object.
    - The name parameter is "Bob".
    - Inside the displayGreeting method, System.out.println(greeting + ", " + name + "!"); is executed.
    - greeting for eveningGreeter is "Good Evening".
    - Output: Good Evening, Bob!
6. The main method finishes.

Complete Output:

Good Morning, Alice!

Good Evening, Bob!

This program demonstrates how objects encapsulate data (greeting field) and behavior (displayGreeting method). Each object (morningGreeter, eveningGreeter) is an independent instance of the MyGreeter class, holding its own greeting value. Methods are invoked on specific objects, affecting or using that object's data. The constructor MyGreeter(String text) is used to initialize the object's state upon creation.