Here are the Java programs for each of your requests, complete with aim, algorithm, pseudocode, code, and output.

---

1. Sum of All Even Numbers Between 1 to n Using For Loop

Aim

To calculate the sum of all even numbers within a given range, from 1 up to n (inclusive), using a for loop.

Algorithm

1. Initialize a variable sum to 0 to store the cumulative sum of even numbers.
2. Define the upper limit n.
3. Use a for loop to iterate through numbers starting from 1 up to n.
4. Inside the loop, for each number, check if it is even using the modulo operator (%). A number is even if number % 2 == 0.
5. If the number is even, add it to the sum.
6. After the loop finishes, print the final sum.

Pseudocode

```
FUNCTION findSumOfEvenNumbers(n)
  sum = 0
  FOR i FROM 1 TO n
    IF i MOD 2 == 0 THEN
      sum = sum + i
```

```
    END IF
  END FOR
  RETURN sum
END FUNCTION

MAIN
  n = 10 // Example upper limit
  result = findSumOfEvenNumbers(n)
  PRINT "Sum of even numbers up to " + n + " is: " + result
END MAIN
```

Code

Java

```java
public class EvenSumForLoop {
  public static void main(String[] args) {
    int n = 10; // The upper limit
    int sum = 0; // Variable to store the sum

    System.out.println("Finding sum of even numbers from 1 to " + n + " using a for loop:");

    for (int i = 1; i <= n; i++) {
      if (i % 2 == 0) { // Check if the number is even
        sum += i; // Add even number to sum
        System.out.println("Adding even number: " + i + ", Current sum: " + sum);
      }
    }

    System.out.println("Final sum of even numbers up to " + n + " is: " + sum);
  }
}
```

Output

```
Finding sum of even numbers from 1 to 10 using a for loop:
Adding even number: 2, Current sum: 2
Adding even number: 4, Current sum: 6
Adding even number: 6, Current sum: 12
Adding even number: 8, Current sum: 20
Adding even number: 10, Current sum: 30
Final sum of even numbers up to 10 is: 30
```

---

2. Sum of All Even Numbers Between 1 to n Using For-Each

Aim

To calculate the sum of all even numbers within a given range, from 1 up to n (inclusive), utilizing an enhanced for-each loop. Since for-each loops work on collections or arrays, we will first populate a list with numbers from 1 to n.

Algorithm

1. Initialize a variable sum to 0.
2. Define the upper limit n.
3. Create an ArrayList to store numbers from 1 to n.
4. Use a standard for loop to populate this ArrayList with integers from 1 to n.
5. Use an enhanced for-each loop to iterate through each number in the populated ArrayList.
6. Inside the for-each loop, check if the current number is even.
7. If it's even, add it to the sum.
8. After the for-each loop completes, print the final sum.

Pseudocode

```
FUNCTION findSumOfEvenNumbersForEach(n)
    sum = 0
    numbersList = NEW ArrayList()

    // Populate the list
    FOR i FROM 1 TO n
        numbersList.add(i)
    END FOR

    // Iterate using for-each
    FOR EACH number IN numbersList
        IF number MOD 2 == 0 THEN
            sum = sum + number
        END IF
    END FOR
    RETURN sum
END FUNCTION

MAIN
    n = 10 // Example upper limit
    result = findSumOfEvenNumbersForEach(n)
    PRINT "Sum of even numbers up to " + n + " is: " + result
END MAIN
```

Code

Java

```java
import java.util.ArrayList;
import java.util.List;

public class EvenSumForEach {
    public static void main(String[] args) {
        int n = 10; // The upper limit
        int sum = 0; // Variable to store the sum

        // Create a list to hold numbers from 1 to n
        List<Integer> numbers = new ArrayList<>();
        for (int i = 1; i <= n; i++) {
            numbers.add(i);
        }

        System.out.println("Finding sum of even numbers from 1 to " + n + " using a for-each loop:");

        // Use a for-each loop to iterate through the list
        for (int number : numbers) {
            if (number % 2 == 0) { // Check if the number is even
                sum += number; // Add even number to sum
                System.out.println("Adding even number: " + number + ", Current sum: " + sum);
            }
        }

        System.out.println("Final sum of even numbers up to " + n + " is: " + sum);
    }
}
```

Output

```
Finding sum of even numbers from 1 to 10 using a for-each loop:
Adding even number: 2, Current sum: 2
Adding even number: 4, Current sum: 6
Adding even number: 6, Current sum: 12
Adding even number: 8, Current sum: 20
```

```
Adding even number: 10, Current sum: 30
Final sum of even numbers up to 10 is: 30
```

---

3. Reverse the Given Digits Using While Loop

Aim

To reverse the digits of a given non-negative integer using a while loop. For example, if the input is 123, the output should be 321.

Algorithm

1. Initialize a variable reversedNumber to 0.
2. Take the input number (the number to be reversed).
3. Use a while loop that continues as long as number is greater than 0.
4. Inside the loop:
    - Get the **last digit** of number using the modulo operator: digit = number % 10.
    - Build the reversedNumber by multiplying it by 10 and adding the digit: reversedNumber = reversedNumber * 10 + digit.
    - Remove the last digit from number by integer division: number = number / 10.
5. Once the loop finishes (when number becomes 0), reversedNumber will hold the reversed digits.
6. Print the reversedNumber.

Pseudocode

```
FUNCTION reverseDigits(number)
```

```
    reversedNumber = 0
    WHILE number > 0
        digit = number MOD 10
        reversedNumber = reversedNumber * 10 + digit
        number = number DIV 10
    END WHILE
    RETURN reversedNumber
END FUNCTION

MAIN
    num = 12345 // Example number
    reversed = reverseDigits(num)
    PRINT "Original number: " + num
    PRINT "Reversed number: " + reversed
END MAIN
```

Code

Java

```
public class ReverseDigitsWhileLoop {
    public static void main(String[] args) {
        int number = 12345; // The number to be reversed
        int reversedNumber = 0; // Variable to store the reversed number
        int originalNumber = number; // Store original for printing

        System.out.println("Original number: " + originalNumber);

        // Use a while loop to reverse the digits
        while (number > 0) {
            int digit = number % 10; // Get the last digit
            reversedNumber = reversedNumber * 10 + digit; // Append the digit to reversedNumber
            number = number / 10; // Remove the last digit from the original number
            System.out.println("Processing: digit=" + digit + ", reversedNumber=" + reversedNumber +
", remaining number=" + number);
        }
```

```
        System.out.println("Reversed number: " + reversedNumber);
    }
}
```

Output

```
Original number: 12345
Processing: digit=5, reversedNumber=5, remaining number=1234
Processing: digit=4, reversedNumber=54, remaining number=123
Processing: digit=3, reversedNumber=543, remaining number=12
Processing: digit=2, reversedNumber=5432, remaining number=1
Processing: digit=1, reversedNumber=54321, remaining number=0
Reversed number: 54321
```

---

4. Check Whether a Given Number is Prime or Not Using Do-While

Aim

To determine if a given positive integer is a prime number using a do-while loop. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Algorithm

1. Take the input number.
2. Handle special cases:
    - If number is less than or equal to 1, it's not prime.

    - If number is 2, it's prime.
3. Initialize a boolean variable isPrime to true.
4. Initialize a counter variable i to 2.
5. Use a do-while loop to check for divisors:
    - Inside the loop, check if number is divisible by i (i.e., number % i == 0).
    - If it is divisible, set isPrime to false and break the loop (as it's not prime).
    - Increment i by 1.
    - The loop condition should be i * i <= number (or i <= Math.sqrt(number)) to optimize checks, as divisors always come in pairs, and one of the pair must be less than or equal to the square root of the number. The do-while loop will execute at least once.
6. After the loop, print whether number is prime based on the isPrime flag.

Pseudocode

```
FUNCTION isPrime(number)
  IF number <= 1 THEN
    RETURN FALSE
  END IF
  IF number == 2 THEN
    RETURN TRUE
  END IF

  isPrimeFlag = TRUE
  i = 2
  DO
    IF number MOD i == 0 THEN
      isPrimeFlag = FALSE
      BREAK // Exit loop early
    END IF
    i = i + 1
  WHILE i * i <= number // Optimized check up to square root

  RETURN isPrimeFlag
END FUNCTION

MAIN
```

```
  num1 = 7 // Example prime number
  num2 = 10 // Example non-prime number

  PRINT num1 + " is prime: " + isPrime(num1)
  PRINT num2 + " is prime: " + isPrime(num2)
END MAIN
```

Code

Java

```
public class PrimeCheckerDoWhile {
  public static void main(String[] args) {
    checkPrime(7);  // Prime number
    checkPrime(10); // Non-prime number
    checkPrime(2);  // Edge case: Prime
    checkPrime(1);  // Edge case: Not prime
    checkPrime(0);  // Edge case: Not prime
    checkPrime(29); // Another prime
  }

  public static void checkPrime(int number) {
    boolean isPrime = true;

    if (number <= 1) {
      isPrime = false; // Numbers less than or equal to 1 are not prime
    } else if (number == 2) {
      isPrime = true; // 2 is the only even prime number
    } else {
      int i = 2;
      do {
        if (number % i == 0) {
          isPrime = false; // Found a divisor, not prime
          break; // Exit loop early
        }
        i++;
```

```
        } while (i * i <= number); // Optimized condition: check up to sqrt(number)
    }

    if (isPrime) {
        System.out.println(number + " is a prime number.");
    } else {
        System.out.println(number + " is NOT a prime number.");
    }
  }
}
```

Output

```
7 is a prime number.
10 is NOT a prime number.
2 is a prime number.
1 is NOT a prime number.
0 is NOT a prime number.
29 is a prime number.
```

---

5. Dog Class with Parameterized Constructor

Aim

To create a Java class called Dog with instance variables for name and color. The main goal is to implement a **parameterized constructor** that takes both name and color as arguments and uses them to initialize the instance variables. Finally, the program will demonstrate this by creating a Dog object and printing its initialized attributes.

Algorithm

1. Define a class named Dog.
2. Declare two instance variables: String name and String color.
3. Create a public constructor Dog(String name, String color).
4. Inside the constructor, use this.name = name; and this.color = color; to assign the passed parameters to the instance variables.
5. In the main method, create an instance of the Dog class, e.g., myDog, passing "Buddy" and "Golden" to its constructor.
6. Access and print the name and color variables of the myDog object.

Pseudocode

```
CLASS Dog
    name: String
    color: String

    CONSTRUCTOR(nameParam, colorParam)
        this.name = nameParam
        this.color = colorParam

END CLASS

MAIN
    myDog = NEW Dog("Buddy", "Golden")
    PRINT "Dog's Name: " + myDog.name
    PRINT "Dog's Color: " + myDog.color
END MAIN
```

Code

Java

```java
class Dog {
    String name;
    String color;

    // Parameterized constructor
    public Dog(String name, String color) {
        this.name = name;     // Initialize the name instance variable
        this.color = color;   // Initialize the color instance variable
        System.out.println("A new Dog object has been created using the parameterized constructor.");
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an instance of the Dog class using the parameterized constructor
        Dog myDog = new Dog("Buddy", "Golden");

        // Print the values of the instance variables
        System.out.println("Dog Details:");
        System.out.println("Name: " + myDog.name);
        System.out.println("Color: " + myDog.color);
    }
}
```

Output

```
A new Dog object has been created using the parameterized constructor.
Dog Details:
Name: Buddy
```

Color: Golden

---

6. Book Class with Multiple Constructors

Aim

To create a Java class called Book with title, author, and price as instance variables. The program will demonstrate the use of different types of constructors: a **default constructor** and **two parameterized constructors** (one taking title and author, and another taking title, author, and price). It will then instantiate Book objects using each constructor and print their attribute values.

Algorithm

1. Define a class named Book.
2. Declare instance variables: String title, String author, and double price.
3. Implement a **default constructor** (public Book()) that initializes variables to default values (e.g., null for strings, 0.0 for double, or "N/A").
4. Implement the first **parameterized constructor** (public Book(String title, String author)) that takes title and author as parameters and initializes them, setting price to a default value like 0.0.
5. Implement the second **parameterized constructor** (public Book(String title, String author, double price)) that takes title, author, and price as parameters and initializes all three instance variables.
6. In the main method:
    - Create a Book object using the default constructor.
    - Create another Book object using the title and author parameterized constructor.
    - Create a third Book object using the title, author, and price parameterized constructor.
    - Print the details (title, author, price) for each of these Book objects.

Pseudocode

```
CLASS Book
    title: String
    author: String
    price: Double

    // Default Constructor
    CONSTRUCTOR()
        this.title = "Unknown"
        this.author = "Unknown"
        this.price = 0.0

    // Parameterized Constructor (title, author)
    CONSTRUCTOR(titleParam, authorParam)
        this.title = titleParam
        this.author = authorParam
        this.price = 0.0 // Default price

    // Parameterized Constructor (title, author, price)
    CONSTRUCTOR(titleParam, authorParam, priceParam)
        this.title = titleParam
        this.author = authorParam
        this.price = priceParam

    METHOD printBookDetails()
        PRINT "Title: " + this.title + ", Author: " + this.author + ", Price: $" + this.price

END CLASS

MAIN
    book1 = NEW Book()
    book2 = NEW Book("1984", "George Orwell")
    book3 = NEW Book("The Hobbit", "J.R.R. Tolkien", 15.99)

    PRINT "Book 1 (Default):"
    book1.printBookDetails()
```

```
    PRINT "Book 2 (Title, Author):"
    book2.printBookDetails()

    PRINT "Book 3 (Title, Author, Price):"
    book3.printBookDetails()
END MAIN
```

Code

Java

```
class Book {
    String title;
    String author;
    double price;

    // Default constructor
    public Book() {
        this.title = "Untitled";
        this.author = "Unknown";
        this.price = 0.0;
        System.out.println("Book created with default constructor.");
    }

    // Parameterized constructor: takes title and author
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.price = 0.0; // Price defaults to 0.0
        System.out.println("Book created with title and author constructor.");
    }

    // Parameterized constructor: takes title, author, and price
    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
```

```
        this.price = price;
        System.out.println("Book created with title, author, and price constructor.");
    }

    // Method to print book details
    public void printBookDetails() {
        System.out.println("  Title: " + title);
        System.out.println("  Author: " + author);
        System.out.println("  Price: $" + String.format("%.2f", price));
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println("--- Creating books ---");

        // Create book using the default constructor
        Book book1 = new Book();
        System.out.println("\nDetails for Book 1:");
        book1.printBookDetails();

        // Create book using the constructor with title and author
        Book book2 = new Book("1984", "George Orwell");
        System.out.println("\nDetails for Book 2:");
        book2.printBookDetails();

        // Create book using the constructor with title, author, and price
        Book book3 = new Book("The Hobbit", "J.R.R. Tolkien", 15.99);
        System.out.println("\nDetails for Book 3:");
        book3.printBookDetails();
    }
}
```

Output

```
--- Creating books ---
```

```
Book created with default constructor.
Book created with title and author constructor.
Book created with title, author, and price constructor.

Details for Book 1:
 Title: Untitled
 Author: Unknown
 Price: $0.00

Details for Book 2:
 Title: 1984
 Author: George Orwell
 Price: $0.00

Details for Book 3:
 Title: The Hobbit
 Author: J.R.R. Tolkien
 Price: $15.99
```

---

7. Print Various Star Patterns

Aim

To print three different star patterns using nested loops in Java:
i) An increasing right-angled triangle.
ii) A decreasing right-angled triangle.
iii) A diamond (or sandglass) pattern.

Algorithm

i) Increasing Right-Angled Triangle

1. Determine the number of rows (e.g., n).
2. Use an outer for loop to iterate from row = 1 to n (for each line).

3. Use an inner for loop to iterate from col = 1 to row.
4. Inside the inner loop, print a star (*).
5. After the inner loop completes (end of a row), print a new line character.

ii) Decreasing Right-Angled Triangle

1. Determine the number of rows (e.g., n).
2. Use an outer for loop to iterate from row = n down to 1.
3. Use an inner for loop to iterate from col = 1 to row.
4. Inside the inner loop, print a star (*).
5. After the inner loop completes, print a new line character.

iii) Diamond Pattern

1. This pattern can be seen as an increasing triangle followed by a decreasing triangle (without repeating the middle row).
2. Determine the number of rows for the top half (e.g., n).
3. **Top Half (Increasing Triangle with spaces):**
    - Outer loop for row = 1 to n.
    - Inner loop for spaces: print n - row spaces.
    - Inner loop for stars: print row stars.
    - Newline.
4. **Bottom Half (Decreasing Triangle with spaces):**
    - Outer loop for row = n - 1 down to 1.
    - Inner loop for spaces: print n - row spaces.
    - Inner loop for stars: print row stars.
    - Newline.

Self-correction: The request pattern for 'c' is just increasing then decreasing like a triangle where the largest row is printed once, then the next row down from that, etc. So it's simply combining patterns i) and ii) but without the leading spaces, making it a "sandglass" type pattern aligned to the left.

Let's refine pattern c) algorithm based on the example:
iii) Combined Increasing and Decreasing (Left-aligned Sandglass)

1. Determine the maximum number of stars in the middle row (e.g., n).
2. **Increasing part:**
    - Outer loop for row = 1 to n.
    - Inner loop for col = 1 to row.
    - Print *.
    - Newline.
3. **Decreasing part:**
    - Outer loop for row = n - 1 down to 1.
    - Inner loop for col = 1 to row.
    - Print *.

- Newline.

Pseudocode

i) Increasing Triangle

```
FUNCTION printIncreasingTriangle(n)
    FOR row FROM 1 TO n
        FOR col FROM 1 TO row
            PRINT "*"
        END FOR
        PRINT NEWLINE
    END FOR
END FUNCTION
```

ii) Decreasing Triangle

```
FUNCTION printDecreasingTriangle(n)
    FOR row FROM n DOWNTO 1
        FOR col FROM 1 TO row
            PRINT "*"
        END FOR
        PRINT NEWLINE
    END FOR
END FUNCTION
```

iii) Combined Increasing/Decreasing (Left-aligned Sandglass)

```
FUNCTION printCombinedTriangle(n)
```

```
    // Increasing part
    FOR row FROM 1 TO n
        FOR col FROM 1 TO row
            PRINT "*"
        END FOR
        PRINT NEWLINE
    END FOR

    // Decreasing part
    FOR row FROM n - 1 DOWNTO 1
        FOR col FROM 1 TO row
            PRINT "*"
        END FOR
        PRINT NEWLINE
    END FOR
END FUNCTION
```

Code

Java

```
public class StarPatterns {
    public static void main(String[] args) {
        int n = 5; // Number of rows for the patterns

        System.out.println("Pattern i): Increasing Right-Angled Triangle");
        printIncreasingTriangle(n);
        System.out.println("\n--------------------\n");

        System.out.println("Pattern ii): Decreasing Right-Angled Triangle");
        printDecreasingTriangle(n);
        System.out.println("\n--------------------\n");

        System.out.println("Pattern c): Combined Increasing and Decreasing (Left-aligned Sandglass)");
        printCombinedTriangle(n);
    }
```

```java
// Pattern i)
public static void printIncreasingTriangle(int n) {
    // Outer loop for rows
    for (int i = 1; i <= n; i++) {
        // Inner loop for columns (prints stars)
        for (int j = 1; j <= i; j++) {
            System.out.print("* "); // Print a star followed by a space
        }
        System.out.println(); // Move to the next line after each row
    }
}

// Pattern ii)
public static void printDecreasingTriangle(int n) {
    // Outer loop for rows (starts from max stars, decreases)
    for (int i = n; i >= 1; i--) {
        // Inner loop for columns (prints stars)
        for (int j = 1; j <= i; j++) {
            System.out.print("* "); // Print a star followed by a space
        }
        System.out.println(); // Move to the next line after each row
    }
}

// Pattern c)
public static void printCombinedTriangle(int n) {
    // Part 1: Increasing triangle
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            System.out.print("* ");
        }
        System.out.println();
    }

    // Part 2: Decreasing triangle (starting from n-1 to avoid repeating the middle row)
    for (int i = n - 1; i >= 1; i--) {
        for (int j = 1; j <= i; j++) {
            System.out.print("* ");
        }
        System.out.println();
    }
}
```

```
}
```

Output

```
Pattern i): Increasing Right-Angled Triangle
* * * * * * * * * * * * * * * --------------------

Pattern ii): Decreasing Right-Angled Triangle
* * * * * * * * * * * * * * * --------------------

Pattern c): Combined Increasing and Decreasing (Left-aligned Sandglass)
* * * * * * * * * * * * * * * * * * * * * * * * * * * ```
```