The dot string response for the below add two numbers java code is

Add two numbers Java code:

```
public class SumOfNumbers1
{
public static void main(String args[])
{
int n1 = 225, n2 = 115, sum;
sum = n1 + n2;
System.out.println("The sum of numbers is: "+sum);
}
```

Properly formatted dot string for the Above add numbers/ sum of numbers code is

```
digraph ControlFlowDiagram {
    node [shape=rectangle];

    Start -> Initialize;
    Initialize -> Calculate;
    Calculate -> Output;
    Output -> End;

    Start [label="Start"];
    Initialize [label="int n1 = 225, n2 = 115, sum;"];
    Calculate [label="sum = n1 + n2;"];
    Output [label="System.out.println(\"The sum of numbers is: \" + sum);"];
    End [label="End"];

    Start [shape=ellipse];
    End [shape=ellipse];
}
```

1. Calculate Sum of Two Numbers

```
int x = 5;
int y = 6;
int sum = x + y;
System.out.println(sum);
```

Properly formatted dot string for the Above add numbers/ sum of numbers code is

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_x [label="int x = 5;"];
    Declare_y [label="int y = 6;"];
    Calculate_sum [label="int sum = x + y;"];
    Print_sum [label="System.out.println(sum);"];
    End [label="End"];
    Start -> Declare_x;
    Declare_x -> Declare_y;
    Declare_y -> Calculate_sum;
    Calculate_sum -> Print_sum;
    Print_sum -> End;
}
```

2.Swap Two Numbers

```
public class SwapNumbers {
    public static void main(String[] args) {
        float first = 1.20f, second = 2.45f;
System.out.println("--Before swap--");
 System.out.println("First number = " + first);
System.out.println("Second number = " + second);
        // Value of first is assigned to temporary
        float temporary = first;
        // Value of second is assigned to first
        first = second;
        // Value of temporary (which contains the initial value of first) is assigned to second
        second = temporary;
        System.out.println("--After swap--");
        System.out.println("First number = " + first);
        System.out.println("Second number = " + second);
    }
}
```

Properly formatted dot string for the Above Swap Two Numbers code is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="float first = 1.20f, second = 2.45f;"];
    Print_before_swap [label="System.out.println(\"--Before swap--\");\nSystem.out.println(\"First number = \" +
first);\nSystem.out.println(\"Second number = \" + second);"];
    Assign_temporary [label="float temporary = first;"];
    Assign_first [label="first = second;"];
    Assign_second [label="second = temporary;"];
    Print_after_swap [label="System.out.println(\"--After swap--\");\nSystem.out.println(\"First number = \" +
first);\nSystem.out.println(\"Second number = \" + second);"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Print_before_swap;
    Print_before_swap -> Assign_temporary;
    Assign_temporary -> Assign_first;
    Assign_first -> Assign_second;
    Assign_second -> Print_after_swap;
    Print_after_swap -> End;
}
```

3. Check if a Number is Even or Odd:

```
import java.util.Scanner;

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

        Scanner reader = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = reader.nextInt();

        if(num % 2 == 0)
            System.out.println(num + " is even");
        else
            System.out.println(num + " is odd");
    }
}
```

Properly formatted dot string for the Above to Check if a Number is Even or Odd:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_reader [label="Scanner reader = new Scanner(System.in);"];
    Print_enter_number [label="System.out.print(\"Enter a number: \");"];
    Read_number [label="int num = reader.nextInt();"];
    If_block [label="if(num % 2 == 0)"];
    If_action [label="System.out.println(num + \" is even\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(num + \" is odd\");"];
    End [label="End"];

    Start -> Declare_reader;
    Declare_reader -> Print_enter_number;
    Print_enter_number -> Read_number;
    Read_number -> If_block;
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

4. Factorial Calculation

```
class FactorialExample{
 public static void main(String args[]){
  int i,fact=1;
  int number=5;//It is the number to calculate factorial
  for(i=1;i<=number;i++){
      fact=fact*i;
  }
  System.out.println("Factorial of "+number+" is: "+fact);
 }
}
```

Properly formatted dot string for the Above Factorial Calculation

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="int i, fact = 1;\nint number = 5;"];
    Initialize_i [label="i = 1;"];
    Loop_condition [label="i <= number;"];
    Update_i [label="i++;"];
    Calculate_factorial [label="fact = fact * i;"];
    Print_result [label="System.out.println(\"Factorial of \" + number + \" is: \" + fact);"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Initialize_i;
    Initialize_i -> Loop_condition;
    Loop_condition -> Calculate_factorial [label="true"];
    Loop_condition -> End [label="false"];
    Calculate_factorial -> Update_i;
    Update_i -> Loop_condition;
    Loop_condition -> Print_result [label="false"];
    Print_result -> End;
}
```

5. Fibonacci Series:

```
class FibonacciExample1{
public static void main(String args[])
{
 int n1=0,n2=1,n3,i,count=10;
 System.out.print(n1+" "+n2);//printing 0 and 1

 for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 }

}}
```

Properly formatted dot string for the Above Fibonacci Series:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="int n1 = 0, n2 = 1, n3, i, count = 10;"];
    Print_initial_numbers [label="System.out.print(n1 + \" \" + n2);"];
    Loop_condition [label="i < count;"];
    Calculate_next_fibonacci [label="n3 = n1 + n2;\nSystem.out.print(\" \" + n3);\nn1 = n2;\nn2 = n3;"];
    Update_i [label="i++"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Print_initial_numbers;
    Print_initial_numbers -> Loop_condition;
    Loop_condition -> Calculate_next_fibonacci [label="true"];
    Loop_condition -> End [label="false"];
    Calculate_next_fibonacci -> Update_i;
    Update_i -> Loop_condition;
    Loop_condition -> End [label="false"];
}
```

6. Prime Number Check

```
// Java Program to demonstrate
// Brute Force Method
// to check if a number is prime
class GFG {
	static boolean isPrime(int n)
	{
		// Corner case
		if (n <= 1)
			return false;

		// Check from 2 to n-1
		for (int i = 2; i < n; i++)
			if (n % i == 0)
				return false;

		return true;
	}

	// Driver Program
	public static void main(String args[])
	{
		if (isPrime(11))
			System.out.println(" true");
		else
			System.out.println(" false");
		if (isPrime(15))
			System.out.println(" true");
		else
			System.out.println(" false");
	}
}
```

Properly formatted dot string for the Above Prime Number Check is:

```
digraph ControlFlowDiagram {
   node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
   Start [label="Start"];
   Declare_isPrime [label="static boolean isPrime(int n)"];
   Check_corner_case [label="if (n <= 1)"];
   Return_false [label="return false;"];
   Loop_condition [label="int i = 2; i < n; i++"];
   Check_divisibility [label="if (n % i == 0)"];
   Return_false_divisible [label="return false;"];
   Return_true [label="return true;"];
   Declare_main [label="public static void main(String args[])"];
   Check_prime_11 [label="if (isPrime(11))"];
   Print_true [label="System.out.println(\" true\");"];
   Check_prime_15 [label="if (isPrime(15))"];
   Print_false [label="System.out.println(\" false\");"];
   End [label="End"];

   Start -> Declare_isPrime;
   Declare_isPrime -> Check_corner_case;
   Check_corner_case -> Return_false [label="true"];
   Check_corner_case -> Loop_condition [label="false"];
   Loop_condition -> Check_divisibility [label="true"];
   Loop_condition -> Return_true [label="false"];
   Check_divisibility -> Return_false_divisible [label="true"];
   Check_divisibility -> Loop_condition [label="false"];
   Return_false_divisible -> End;
   Return_true -> End;
   Start -> Declare_main;
   Declare_main -> Check_prime_11;
   Check_prime_11 -> Print_true [label="true"];
   Check_prime_11 -> Check_prime_15 [label="false"];
   Print_true -> Check_prime_15;
   Check_prime_15 -> Print_false [label="false"];
   Check_prime_15 -> End [label="true"];
   Print_false -> End;
}
```

7. Reverse a String:

```
// java program to reverse a word

import java.io.*;
import java.util.Scanner;

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

                String str= "Geeks", nstr="";
                char ch;
        System.out.print("Original word: ");
        System.out.println("Geeks"); //Example word

        for (int i=0; i<str.length(); i++)
        {
                ch= str.charAt(i); //extracts each character
                nstr= ch+nstr; //adds each character in front of the existing string
        }
        System.out.println("Reversed word: "+ nstr);
        }
}
```

Properly formatted dot string for the Above Reverse a String code is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_variables [label="String str = \"Geeks\", nstr = \"\";"];
    Print_original_word [label="System.out.print(\"Original word: \");\nSystem.out.println(\"Geeks\");"];
    Initialize_loop_variable [label="int i = 0;"];
    Loop_condition [label="i < str.length();"];
    Extract_character [label="char ch = str.charAt(i);"];
    Append_character [label="nstr = ch + nstr;"];
    Update_loop_variable [label="i++;"];
    Print_reversed_word [label="System.out.println(\"Reversed word: \" + nstr);"];
    End [label="End"];

    Start -> Declare_variables;
    Declare_variables -> Print_original_word;
    Print_original_word -> Initialize_loop_variable;
    Initialize_loop_variable -> Loop_condition;
    Loop_condition -> Extract_character [label="true"];
    Loop_condition -> Print_reversed_word [label="false"];
    Extract_character -> Append_character;
    Append_character -> Update_loop_variable;
    Update_loop_variable -> Loop_condition;
    Print_reversed_word -> End;
}
```

8. Calculate Average of Numbers in an Array

```
public class Average {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 15, 20, 25};
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        double average = (double) sum / numbers.length;
        System.out.println("Average: " + average);
    }
}
```

Properly formatted dot string for the Above Calculate Average of Numbers in an Array is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_numbers [label="int[] numbers = {5, 10, 15, 20, 25};"];
    Initialize_sum [label="int sum = 0;"];
    Loop_start [label="for (int num : numbers)"];
    Add_to_sum [label="sum += num;"];
    Calculate_average [label="double average = (double) sum / numbers.length;"];
    Print_average [label="System.out.println(\"Average: \" + average);"];
    End [label="End"];

    Start -> Declare_numbers;
    Declare_numbers -> Initialize_sum;
    Initialize_sum -> Loop_start;
    Loop_start -> Add_to_sum;
    Add_to_sum -> Loop_start;
    Loop_start -> Calculate_average;
    Calculate_average -> Print_average;
    Print_average -> End;
}
```

9. Bubble Sort:

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90};
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
        System.out.println("Sorted array:");
        for (int value : arr) System.out.print(value + " ");
    }
}
```

Properly formatted dot string for the Above Bubble Sort is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {64, 34, 25, 12, 22, 11, 90};\nint n = arr.length;"];
    Initialize_i [label="int i = 0;"];
    Outer_loop_condition [label="i < n-1;"];
    Initialize_j [label="int j = 0;"];
    Inner_loop_condition [label="j < n-i-1;"];
    Check_swap [label="if (arr[j] > arr[j+1])"];
    Perform_swap [label="int temp = arr[j];\narr[j] = arr[j+1];\narr[j+1] = temp;"];
    Update_j [label="j++;"];
    Update_i [label="i++;"];
    Print_sorted_array [label="System.out.println(\"Sorted array:\");\nfor (int value : arr) System.out.print(value + \" \");"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Initialize_i;
    Initialize_i -> Outer_loop_condition;
    Outer_loop_condition -> Initialize_j [label="true"];
    Outer_loop_condition -> End [label="false"];
    Initialize_j -> Inner_loop_condition;
    Inner_loop_condition -> Check_swap [label="true"];
    Inner_loop_condition -> Update_i [label="false"];
    Check_swap -> Perform_swap [label="true"];
    Check_swap -> Update_j [label="false"];
    Perform_swap -> Update_j;
    Update_j -> Inner_loop_condition;
    Update_i -> Outer_loop_condition;
    Outer_loop_condition -> Print_sorted_array [label="false"];
    Print_sorted_array -> End;
}
```

10. Binary Search:

```
public class BinarySearch {
    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 10, 40};
        int target = 10;
        int result = binarySearch(arr, target);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index " + result);
    }
    static int binarySearch(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target)
                return mid;
            if (arr[mid] < target)
                left = mid + 1;
            else
                right = mid - 1;
        }
        return -1;
    }
}
```

Properly formatted dot string for the Above Binary Search is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {2, 3, 4, 10, 40};\nint target = 10;"];
    Call_binarySearch [label="int result = binarySearch(arr, target);"];
    If_block [label="if (arr[mid] == target)"];
    If_action [label="return mid;"];
    Else_block [label="else"];
    Else_action [label="{\n    if (arr[mid] < target)\n        left = mid + 1;\n    else\n        right = mid - 1;\n}"];
    Declare_variables [label="int left = 0, right = arr.length - 1;"];
    Loop_condition [label="left <= right;"];
    Calculate_mid [label="int mid = left + (right - left) / 2;"];
    Return_negative_one [label="return -1;"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Call_binarySearch;
    Call_binarySearch -> Declare_variables;
    Declare_variables -> Loop_condition;
    Loop_condition -> Calculate_mid [label="true"];
    Loop_condition -> If_block [label="false"];
    Calculate_mid -> If_block;
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> Loop_condition;
    Loop_condition -> Return_negative_one [label="false"];
    Return_negative_one -> End;
}
```

11. Linear Search:

```
public class LinearSearch {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        int target = 30;
        int result = linearSearch(arr, target);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index " + result);
    }
    static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target)
                return i;
        }
        return -1;
    }
}
```

Properly formatted dot string for the Above Linear Search is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {10, 20, 30, 40, 50};\nint target = 30;"];
    Call_linearSearch [label="int result = linearSearch(arr, target);"];
    If_block [label="if (result == -1)"];
    If_action [label="System.out.println(\"Element not present\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(\"Element found at index \" + result);"];
    Loop_start [label="for (int i = 0; i < arr.length; i++)"];
    Check_element [label="if (arr[i] == target)"];
    Check_action [label="return i;"];
    Return_negative_one [label="return -1;"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Call_linearSearch;
    Call_linearSearch -> Loop_start;
    Loop_start -> Check_element;
    Check_element -> If_block [label="true"];
    Check_element -> Loop_start [label="false"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
    Check_element -> Check_action;
    Check_action -> Return_negative_one;
    Return_negative_one -> End;
}
```

12. Implement a Stack:

```
import java.util.Stack;
public class StackExample {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        System.out.println("Stack: " + stack);
        System.out.println("Top element: " + stack.peek());
        System.out.println("Popped element: " + stack.pop());
        System.out.println("Stack after pop operation: " + stack);
    }
}
```

Properly formatted dot string for the Above Implement a Stack is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_stack [label="Stack<Integer> stack = new Stack<>();"];
    Push_10 [label="stack.push(10);"];
    Push_20 [label="stack.push(20);"];
    Push_30 [label="stack.push(30);"];
    Print_stack [label="System.out.println(\"Stack: \" + stack);"];
    Print_top_element [label="System.out.println(\"Top element: \" + stack.peek());"];
    Pop_element [label="System.out.println(\"Popped element: \" + stack.pop());"];
    Print_stack_after_pop [label="System.out.println(\"Stack after pop operation: \" + stack);"];
    End [label="End"];

    Start -> Declare_stack;
    Declare_stack -> Push_10;
    Push_10 -> Push_20;
    Push_20 -> Push_30;
    Push_30 -> Print_stack;
    Print_stack -> Print_top_element;
    Print_top_element -> Pop_element;
    Pop_element -> Print_stack_after_pop;
    Print_stack_after_pop -> End;
}
```

13. Implement a Queue:

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueExample {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(10);
        queue.add(20);
        queue.add(30);
        System.out.println("Queue: " + queue);
        System.out.println("Front element: " + queue.peek());
        System.out.println("Removed element: " + queue.remove());
        System.out.println("Queue after remove operation: " + queue);
    }
}
```

Properly formatted dot string for the Above Implement a Queue is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_queue [label="Queue<Integer> queue = new LinkedList<>();"];
    Add_10 [label="queue.add(10);"];
    Add_20 [label="queue.add(20);"];
    Add_30 [label="queue.add(30);"];
    Print_queue [label="System.out.println(\"Queue: \" + queue);"];
    Print_front_element [label="System.out.println(\"Front element: \" + queue.peek());"];
    Remove_element [label="System.out.println(\"Removed element: \" + queue.remove());"];
    Print_queue_after_remove [label="System.out.println(\"Queue after remove operation: \" + queue);"];
    End [label="End"];

    Start -> Declare_queue;
    Declare_queue -> Add_10;
    Add_10 -> Add_20;
    Add_20 -> Add_30;
    Add_30 -> Print_queue;
    Print_queue -> Print_front_element;
    Print_front_element -> Remove_element;
    Remove_element -> Print_queue_after_remove;
    Print_queue_after_remove -> End;
}
```

14. Implement a HashMap:

```
import java.util.HashMap;
public class HashMapExample {
    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("John", 25);
        map.put("Alice", 30);
        map.put("Bob", 28);
        System.out.println("Map: " + map);
        System.out.println("Age of Alice: " + map.get("Alice"));
        System.out.println("Is map empty? " + map.isEmpty());
        System.out.println("Size of map: " + map.size());
    }
}
```

Properly formatted dot string for the Above Implement a HashMap is :

```
 digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_map [label="HashMap<String, Integer> map = new HashMap<>();"];
    Put_John [label="map.put(\"John\", 25);"];
    Put_Alice [label="map.put(\"Alice\", 30);"];
    Put_Bob [label="map.put(\"Bob\", 28);"];
    Print_map [label="System.out.println(\"Map: \" + map);"];
    Print_age_of_Alice [label="System.out.println(\"Age of Alice: \" + map.get(\"Alice\"));"];
    Check_empty [label="System.out.println(\"Is map empty? \" + map.isEmpty());"];
    Print_size [label="System.out.println(\"Size of map: \" + map.size());"];
    End [label="End"];

    Start -> Declare_map;
    Declare_map -> Put_John;
    Put_John -> Put_Alice;
    Put_Alice -> Put_Bob;
    Put_Bob -> Print_map;
    Print_map -> Print_age_of_Alice;
    Print_age_of_Alice -> Check_empty;
    Check_empty -> Print_size;
    Print_size -> End;
}
```

15. Implement a HashSet:

```
import java.util.HashSet;
public class HashSetExample {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();
        set.add("apple");
        set.add("banana");
        set.add("orange");
        System.out.println("Set: " + set);
        System.out.println("Is 'banana' present? " + set.contains("banana"));
        System.out.println("Size of set: " + set.size());
    }
}
```

Properly formatted dot string for the Above Implement a HashSet is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_set [label="HashSet<String> set = new HashSet<>();"];
    Add_apple [label="set.add(\"apple\");"];
    Add_banana [label="set.add(\"banana\");"];
    Add_orange [label="set.add(\"orange\");"];
    Print_set [label="System.out.println(\"Set: \" + set);"];
    Check_banana [label="System.out.println(\"Is 'banana' present? \" + set.contains(\"banana\"));"];
    Print_size [label="System.out.println(\"Size of set: \" + set.size());"];
    End [label="End"];

    Start -> Declare_set;
    Declare_set -> Add_apple;
    Add_apple -> Add_banana;
    Add_banana -> Add_orange;
    Add_orange -> Print_set;
    Print_set -> Check_banana;
    Check_banana -> Print_size;
    Print_size -> End;
}
```

16. Implement an ArrayList:

```
import java.util.ArrayList;
public class ArrayListExample {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("apple");
        list.add("banana");
        list.add("orange");
        System.out.println("List: " + list);
        System.out.println("Element at index 1: " + list.get(1));
        System.out.println("Index of 'banana': " + list.indexOf("banana"));
        list.remove(1);
        System.out.println("List after removing element at index 1: " + list);
    }
}
```

Properly formatted dot string for the Above Implement an ArrayList is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_list [label="ArrayList<String> list = new ArrayList<>();"];
    Add_apple [label="list.add(\"apple\");"];
    Add_banana [label="list.add(\"banana\");"];
    Add_orange [label="list.add(\"orange\");"];
    Print_list [label="System.out.println(\"List: \" + list);"];
    Print_element_at_index_1 [label="System.out.println(\"Element at index 1: \" + list.get(1));"];
    Print_index_of_banana [label="System.out.println(\"Index of 'banana': \" + list.indexOf(\"banana\"));"];
    Remove_element_at_index_1 [label="list.remove(1);"];
    Print_list_after_remove [label="System.out.println(\"List after removing element at index 1: \" + list);"];
    End [label="End"];

    Start -> Declare_list;
    Declare_list -> Add_apple;
    Add_apple -> Add_banana;
    Add_banana -> Add_orange;
    Add_orange -> Print_list;
    Print_list -> Print_element_at_index_1;
    Print_element_at_index_1 -> Print_index_of_banana;
    Print_index_of_banana -> Remove_element_at_index_1;
    Remove_element_at_index_1 -> Print_list_after_remove;
    Print_list_after_remove -> End;
}
```

17. Find Maximum Element in an Array:

```
public class MaxElement {
    public static void main(String[] args) {
        int[] arr = {5, 10, 3, 8, 15};
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > max)
                max = arr[i];
        }
        System.out.println("Maximum element: " + max);
    }
}
```

Properly formatted dot string for the Above Find Maximum Element in an Array is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {5, 10, 3, 8, 15};"];
    Initialize_max [label="int max = arr[0];"];
    Initialize_i [label="int i = 1;"];
    Loop_condition [label="i < arr.length;"];
    Check_max [label="if (arr[i] > max)\n    max = arr[i];"];
    Update_i [label="i++;"];
    Print_max [label="System.out.println(\"Maximum element: \" + max);"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Initialize_max;
    Initialize_max -> Initialize_i;
    Initialize_i -> Loop_condition;
    Loop_condition -> Check_max [label="true"];
    Loop_condition -> Print_max [label="false"];
    Check_max -> Update_i;
    Update_i -> Loop_condition;
    Print_max -> End;
}
```

18. Implement a Singly Linked List:

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class LinkedListExample {
    Node head;
    public static void main(String[] args) {
        LinkedListExample list = new LinkedListExample();
        list.addNode(10);
        list.addNode(20);
        list.addNode(30);
        list.printList();
    }
    void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null)
            head = newNode;
        else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = newNode;
        }
    }
    void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
    }
}
```

Properly formatted dot string for the Above Implement a Singly Linked List is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_head [label="Node head;"];
    Create_list_instance [label="LinkedListExample list = new LinkedListExample();"];
    Add_node_10 [label="list.addNode(10);"];
    Add_node_20 [label="list.addNode(20);"];
    Add_node_30 [label="list.addNode(30);"];
    Print_list [label="list.printList();"];
    End [label="End"];

    Start -> Declare_head;
    Declare_head -> Create_list_instance;
    Create_list_instance -> Add_node_10;
    Add_node_10 -> Add_node_20;
    Add_node_20 -> Add_node_30;
    Add_node_30 -> Print_list;
    Print_list -> End;
}
```

19. Calculate Area of a Circle:

```
public class AreaOfCircle {
    public static void main(String[] args) {
        double radius = 5;
        double area = Math.PI * radius * radius;
        System.out.println("Area of circle: " + area);
    }
}
```

Properly formatted dot string for the Above Calculate Area of a Circle is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_radius [label="double radius = 5;"];
    Calculate_area [label="double area = Math.PI * radius * radius;"];
    Print_area [label="System.out.println(\"Area of circle: \" + area);"];
    End [label="End"];

    Start -> Declare_radius;
    Declare_radius -> Calculate_area;
    Calculate_area -> Print_area;
    Print_area -> End;
}
```

20. Calculate Area and Perimeter of Rectangle:

```
public class Rectangle {
    public static void main(String[] args) {
        double length = 5;
        double width = 3;
        double area = length * width;
        double perimeter = 2 * (length + width);
        System.out.println("Area of rectangle: " + area);
        System.out.println("Perimeter of rectangle: " + perimeter);
    }
}
```

Properly formatted dot string for the Above Calculate Area and Perimeter of Rectangle:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_length [label="double length = 5;"];
    Declare_width [label="double width = 3;"];
    Calculate_area [label="double area = length * width;"];
    Calculate_perimeter [label="double perimeter = 2 * (length + width);"];
    Print_area [label="System.out.println(\"Area of rectangle: \" + area);"];
    Print_perimeter [label="System.out.println(\"Perimeter of rectangle: \" + perimeter);"];
    End [label="End"];

    Start -> Declare_length;
    Declare_length -> Declare_width;
    Declare_width -> Calculate_area;
    Calculate_area -> Calculate_perimeter;
    Calculate_perimeter -> Print_area;
    Print_area -> Print_perimeter;
    Print_perimeter -> End;
}
```

21. Calculate Area and Circumference of Circle:

```
public class Circle {
    public static void main(String[] args) {
        double radius = 7;
        double area = Math.PI * radius * radius;
        double circumference = 2 * Math.PI * radius;
        System.out.println("Area of circle: " + area);
        System.out.println("Circumference of circle: " + circumference);
    }
}
```

Properly formatted dot string for the Above Calculate Area and Circumference of Circle is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_radius [label="double radius = 7;"];
    Calculate_area [label="double area = Math.PI * radius * radius;"];
    Calculate_circumference [label="double circumference = 2 * Math.PI * radius;"];
    Print_area [label="System.out.println(\"Area of circle: \" + area);"];
    Print_circumference [label="System.out.println(\"Circumference of circle: \" + circumference);"];
    End [label="End"];

    Start -> Declare_radius;
    Declare_radius -> Calculate_area;
    Calculate_area -> Calculate_circumference;
    Calculate_circumference -> Print_area;
    Print_area -> Print_circumference;
    Print_circumference -> End;
}
```

22. Check Leap Year:

```
public class LeapYear {
    public static void main(String[] args) {
        int year = 2024;
        if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
            System.out.println(year + " is a leap year.");
        else
            System.out.println(year + " is not a leap year.");
    }
}
```

Properly formatted dot string for the Above Check Leap Year is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_year [label="int year = 2024;"];
    If_block [label="if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))"];
    If_action [label="System.out.println(year + \" is a leap year.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(year + \" is not a leap year.\");"];
    End [label="End"];

    Start -> Declare_year;
    Declare_year -> If_block;
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

23. Reverse an Array:

```
public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int n = arr.length;
        for (int i = 0; i < n / 2; i++) {
            int temp = arr[i];
            arr[i] = arr[n - i - 1];
            arr[n - i - 1] = temp;
        }
        System.out.println("Reversed array:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

Properly formatted dot string for the Above Reverse an Array is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_array [label="int[] arr = {1, 2, 3, 4, 5};\nint n = arr.length;"];
    Loop_condition [label="for (int i = 0; i < n / 2; i++)"];
    Swap_elements [label="Swap elements"];
    Print_reversed_array [label="Print reversed array"];
    End [label="End"];

    Start -> Declare_array;
    Declare_array -> Loop_condition;
    Loop_condition -> Swap_elements [label="true"];
    Swap_elements -> Loop_condition;
    Loop_condition -> Print_reversed_array [label="false"];
    Print_reversed_array -> End;
}
```

24. Calculate Power of a Number:

```
public class Power {
    public static void main(String[] args) {
        int base = 3;
        int exponent = 4;
        long result = 1;
        while (exponent != 0) {
            result *= base;
            --exponent;
        }
        System.out.println("Result: " + result);
    }
}
```

Properly formatted dot string for the Above Calculate Power of a Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_base [label="int base = 3;"];
    Declare_exponent [label="int exponent = 4;"];
    Initialize_result [label="long result = 1;"];
    Loop_condition [label="while (exponent != 0)"];
    Update_result [label="result *= base;\n--exponent;"];
    Print_result [label="System.out.println(\"Result: \" + result);"];
    End [label="End"];

    Start -> Declare_base;
    Declare_base -> Declare_exponent;
    Declare_exponent -> Initialize_result;
    Initialize_result -> Loop_condition;
    Loop_condition -> Update_result [label="true"];
    Update_result -> Loop_condition;
    Loop_condition -> Print_result [label="false"];
    Print_result -> End;
}
```

25. Generate Random Numbers:

```
import java.util.Random;
public class RandomNumbers {
    public static void main(String[] args) {
        Random rand = new Random();
        int randomInt = rand.nextInt(100); // Generate random integer between 0 and 99
        double randomDouble = rand.nextDouble(); // Generate random double between 0.0 and 1.0
        System.out.println("Random integer: " + randomInt);
        System.out.println("Random double: " + randomDouble);
    }
}
```

Properly formatted dot string for the Above Generate Random Numbers is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_rand [label="Random rand = new Random();"];
    Generate_randomInt [label="int randomInt = rand.nextInt(100); // Generate random integer between 0 and 99"];
    Generate_randomDouble [label="double randomDouble = rand.nextDouble(); // Generate random double between 0.0 and 1.0"];
    Print_randomInt [label="System.out.println(\"Random integer: \" + randomInt);"];
    Print_randomDouble [label="System.out.println(\"Random double: \" + randomDouble);"];
    End [label="End"];

    Start -> Declare_rand;
    Declare_rand -> Generate_randomInt;
    Generate_randomInt -> Generate_randomDouble;
    Generate_randomDouble -> Print_randomInt;
    Print_randomInt -> Print_randomDouble;
    Print_randomDouble -> End;
}
```

26. Print Fibonacci Series Using Recursion:

```
public class FibonacciRecursion {
    public static void main(String[] args) {
        int n = 10;
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }
    static int fibonacci(int n) {
        if (n <= 1)
            return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Properly formatted dot string for the Above Print Fibonacci Series Using Recursion is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_n [label="int n = 10;"];
    Loop_condition [label="for (int i = 0; i < n; i++)"];
    Call_fibonacci [label="fibonacci(i)"];
    Check_base_case [label="if (n <= 1)\n    return n;"];
    Recursive_call_1 [label="fibonacci(n - 1)"];
    Recursive_call_2 [label="fibonacci(n - 2)"];
    Return_value [label="return fibonacci(n - 1) + fibonacci(n - 2);"];
    Print_fibonacci [label="System.out.print(fibonacci(i) + \" \");"];
    End [label="End"];
    Start -> Declare_n;
    Declare_n -> Loop_condition;
    Loop_condition -> Call_fibonacci [label="true"];
    Call_fibonacci -> Check_base_case;
    Check_base_case -> Return_value [label="n <= 1"];
    Check_base_case -> Recursive_call_1 [label="n > 1"];
    Recursive_call_1 -> Return_value;
    Recursive_call_2 -> Return_value;
    Return_value -> Print_fibonacci;
    Print_fibonacci -> Loop_condition;
    Loop_condition -> End [label="i >= n"];
}
```

27. Find GCD (Greatest Common Divisor):

```
public class GCD {
    public static void main(String[] args) {
        int num1 = 12, num2 = 18;
        while (num1 != num2) {
            if (num1 > num2)
                num1 -= num2;
            else
                num2 -= num1;
        }
        System.out.println("GCD: " + num1);
    }
}
```

Properly formatted dot string for the Above Find GCD is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num1 [label="int num1 = 12;"];
    Declare_num2 [label="int num2 = 18;"];
    Loop_condition [label="while (num1 != num2)"];
    Check_num1_greater [label="if (num1 > num2)\n    num1 -= num2;"];
    Check_num2_greater [label="else\n    num2 -= num1;"];
    Print_GCD [label="System.out.println(\"GCD: \" + num1);"];
    End [label="End"];

    Start -> Declare_num1;
    Declare_num1 -> Declare_num2;
    Declare_num2 -> Loop_condition;
    Loop_condition -> Check_num1_greater [label="true"];
    Check_num1_greater -> Loop_condition;
    Loop_condition -> Check_num2_greater [label="false"];
    Check_num2_greater -> Loop_condition;
    Loop_condition -> Print_GCD [label="num1 == num2"];
    Print_GCD -> End;
}
```

28. Count Number of Words in a Sentence:

```
public class WordCount {
    public static void main(String[] args) {
        String sentence = "Java programming is fun";
        int count = 1;
        for (int i = 0; i < sentence.length(); i++) {
            if (sentence.charAt(i) == ' ')
                count++;
        }
        System.out.println("Number of words: " + count);
    }
}
```

Properly formatted dot string for the Above Count Number of Words in a Sentence is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_sentence [label="String sentence = \"Java programming is fun\";"];
    Initialize_count [label="int count = 1;"];
    Loop_condition [label="for (int i = 0; i < sentence.length(); i++)"];
    Check_space [label="if (sentence.charAt(i) == ' ')\n    count++;"];
    Print_count [label="System.out.println(\"Number of words: \" + count);"];
    End [label="End"];

    Start -> Declare_sentence;
    Declare_sentence -> Initialize_count;
    Initialize_count -> Loop_condition;
    Loop_condition -> Check_space [label="true"];
    Check_space -> Loop_condition;
    Loop_condition -> Print_count [label="i == sentence.length()"];
    Print_count -> End;
}
```

29. Check Palindrome:

```
public class Palindrome {
    public static void main(String[] args) {
        String str = "radar";
        boolean isPalindrome = true;
        for (int i = 0; i < str.length() / 2; i++) {
            if (str.charAt(i) != str.charAt(str.length() - i - 1)) {
                isPalindrome = false;
                break;
            }
        }
        if (isPalindrome)
            System.out.println(str + " is a palindrome.");
        else
            System.out.println(str + " is not a palindrome.");
    }
}
```

Properly formatted dot string for the Above Check Palindrome is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_str [label="String str = \"radar\";"];
    Initialize_isPalindrome [label="boolean isPalindrome = true;"];
    Loop_condition [label="for (int i = 0; i < str.length() / 2; i++)"];
    Check_characters [label="if (str.charAt(i) != str.charAt(str.length() - i - 1)) {\n    isPalindrome = false;\n    break;\n}"];
    If_block [label="if (isPalindrome)"];
    If_action [label="System.out.println(str + \" is a palindrome.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(str + \" is not a palindrome.\");"];
    End [label="End"];

    Start -> Declare_str;
    Declare_str -> Initialize_isPalindrome;
    Initialize_isPalindrome -> Loop_condition;
    Loop_condition -> Check_characters [label="true"];
    Check_characters -> Loop_condition;
    Loop_condition -> If_block [label="i == str.length() / 2"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

30. Convert Decimal to Binary:

```
public class DecimalToBinary {
    public static void main(String[] args) {
        int decimal = 10;
        StringBuilder binary = new StringBuilder();
        while (decimal > 0) {
            binary.insert(0, decimal % 2);
            decimal /= 2;
        }
        System.out.println("Binary representation: " + binary);
    }
}
```

Properly formatted dot string for the Above Convert Decimal to Binary is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_decimal [label="int decimal = 10;"];
    Initialize_binary [label="StringBuilder binary = new StringBuilder();"];
    Loop_condition [label="while (decimal > 0)"];
    Calculate_binary_digit [label="binary.insert(0, decimal % 2);\ndecimal /= 2;"];
    Print_binary [label="System.out.println(\"Binary representation: \" + binary);"];
    End [label="End"];

    Start -> Declare_decimal;
    Declare_decimal -> Initialize_binary;
    Initialize_binary -> Loop_condition;
    Loop_condition -> Calculate_binary_digit [label="true"];
    Calculate_binary_digit -> Loop_condition;
    Loop_condition -> Print_binary [label="decimal <= 0"];
    Print_binary -> End;
}
```

31. Convert Binary to Decimal:

```
public class BinaryToDecimal {
    public static void main(String[] args) {
        String binary = "1010";
        int decimal = Integer.parseInt(binary, 2);
        System.out.println("Decimal representation: " + decimal);
    }
}
```

Properly formatted dot string for the Above Convert Binary to Decimal is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_binary [label="String binary = \"1010\";"];
    Convert_to_decimal [label="int decimal = Integer.parseInt(binary, 2);"];
    Print_decimal [label="System.out.println(\"Decimal representation: \" + decimal);"];
    End [label="End"];

    Start -> Declare_binary;
    Declare_binary -> Convert_to_decimal;
    Convert_to_decimal -> Print_decimal;
    Print_decimal -> End;
}
```

32. Check Armstrong Number:

```
public class ArmstrongNumber {
    public static void main(String[] args) {
        int num = 153;
        int originalNum = num;
        int result = 0;
        while (num != 0) {
            int remainder = num % 10;
            result += Math.pow(remainder, 3);
            num /= 10;
        }
        if (result == originalNum)
            System.out.println(originalNum + " is an Armstrong number.");
        else
            System.out.println(originalNum + " is not an Armstrong number.");
    }
}
```

Properly formatted dot string for the Above Check Armstrong Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num [label="int num = 153;"];
    Assign_originalNum [label="int originalNum = num;"];
    Initialize_result [label="int result = 0;"];
    Loop_condition [label="while (num != 0)"];
    Calculate_remainder [label="int remainder = num % 10;"];
    Calculate_result [label="result += Math.pow(remainder, 3);"];
    Update_num [label="num /= 10;"];
    If_block [label="if (result == originalNum)"];
    If_action [label="System.out.println(originalNum + \" is an Armstrong number.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(originalNum + \" is not an Armstrong number.\");"];
    End [label="End"];

    Start -> Declare_num;
    Declare_num -> Assign_originalNum;
    Assign_originalNum -> Initialize_result;
    Initialize_result -> Loop_condition;
    Loop_condition -> Calculate_remainder [label="true"];
    Calculate_remainder -> Calculate_result;
    Calculate_result -> Update_num;
    Update_num -> Loop_condition;
    Loop_condition -> If_block [label="num == 0"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

33. Find Factorial Using Recursion:

```
public class FactorialRecursion {
    public static void main(String[] args) {
        int num = 5;
        System.out.println("Factorial of " + num + " = " + factorial(num));
    }
    static int factorial(int n) {
        if (n == 0)
            return 1;
        else
            return n * factorial(n - 1);
    }
}
```

Properly formatted dot string for the Above Find Factorial Using Recursion is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num [label="int num = 5;"];
    Call_factorial [label="factorial(num)"];
    Check_base_case [label="if (n == 0)\n    return 1;"];
    Recursive_call [label="return n * factorial(n - 1);"];
    Print_factorial [label="System.out.println(\"Factorial of \" + num + \" = \" + factorial(num));"];
    End [label="End"];

    Start -> Declare_num;
    Declare_num -> Call_factorial;
    Call_factorial -> Check_base_case;
    Check_base_case -> Recursive_call [label="n != 0"];
    Recursive_call -> Check_base_case;
    Check_base_case -> Print_factorial;
    Print_factorial -> End;
}
```

34. Check Perfect Number:

```
public class PerfectNumber {
    public static void main(String[] args) {
        int num = 28;
        int sum = 0;
        for (int i = 1; i <= num / 2; i++) {
            if (num % i == 0)
                sum += i;
        }
        if (sum == num)
            System.out.println(num + " is a perfect number.");
        else
            System.out.println(num + " is not a perfect number.");
```

Properly formatted dot string for the Above Check Perfect Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_num [label="int num = 28;"];
    Initialize_sum [label="int sum = 0;"];
    Loop_condition [label="for (int i = 1; i <= num / 2; i++)"];
    Check_divisibility [label="if (num % i == 0)\n    sum += i;"];
    If_block [label="if (sum == num)"];
    If_action [label="System.out.println(num + \" is a perfect number.\");"];
    Else_block [label="else"];
    Else_action [label="System.out.println(num + \" is not a perfect number.\");"];
    End [label="End"];
    Start -> Declare_num;
    Declare_num -> Initialize_sum;
    Initialize_sum -> Loop_condition;
    Loop_condition -> Check_divisibility [label="true"];
    Check_divisibility -> Loop_condition;
    Loop_condition -> If_block [label="i > num / 2"];
    If_block -> If_action;
    If_action -> End;
    If_block -> Else_block;
    Else_block -> Else_action;
    Else_action -> End;
}
```

35. Implementing a Binary Search Tree (BST)

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    private TreeNode root;

    public BinarySearchTree() {
        this.root = null;
    }

    public void insert(int val) {
        this.root = insertNode(this.root, val);
    }

    private TreeNode insertNode(TreeNode root, int val) {
        if (root == null) {
            return new TreeNode(val);
        }
        if (val < root.val) {
            root.left = insertNode(root.left, val);
        } else if (val > root.val) {
            root.right = insertNode(root.right, val);
        }
        return root;
    }

    public boolean search(int val) {
        return searchNode(this.root, val);
    }

    private boolean searchNode(TreeNode root, int val) {
        if (root == null) {
            return false;
        }
        if (root.val == val) {
            return true;
        }
        if (val < root.val) {
            return searchNode(root.left, val);
        } else {
            return searchNode(root.right, val);
        }
    }

    public void inorderTraversal() {
        performInorderTraversal(this.root);
    }

    private void performInorderTraversal(TreeNode root) {
        if (root != null) {
            performInorderTraversal(root.left);
            System.out.print(root.val + " ");
            performInorderTraversal(root.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        bst.insert(50);
        bst.insert(30);
        bst.insert(70);
        bst.insert(20);
        bst.insert(40);
        bst.insert(60);
        bst.insert(80);

        System.out.println("Inorder Traversal:");
        bst.inorderTraversal();

        int searchKey = 40;
        System.out.println("\nIs " + searchKey + " present in BST? " + bst.search(searchKey));
    }
}
```

Properly formatted dot string for the Above Implementing a Binary Search Tree is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_bst [label="BinarySearchTree bst = new BinarySearchTree();"];
    Insert_50 [label="bst.insert(50);"];
    Insert_30 [label="bst.insert(30);"];
    Insert_70 [label="bst.insert(70);"];
    Insert_20 [label="bst.insert(20);"];
    Insert_40 [label="bst.insert(40);"];
    Insert_60 [label="bst.insert(60);"];
    Insert_80 [label="bst.insert(80);"];
    Inorder_traversal [label="System.out.println(\"Inorder Traversal:\");\nbst.inorderTraversal();"];
    Declare_searchKey [label="int searchKey = 40;"];
    Search_40 [label="System.out.println(\"\\nIs \" + searchKey + \" present in BST? \" + bst.search(searchKey));"];
    End [label="End"];

    Start -> Declare_bst;
    Declare_bst -> Insert_50;
    Insert_50 -> Insert_30;
    Insert_30 -> Insert_70;
    Insert_70 -> Insert_20;
    Insert_20 -> Insert_40;
    Insert_40 -> Insert_60;
    Insert_60 -> Insert_80;
    Insert_80 -> Inorder_traversal;
    Inorder_traversal -> Declare_searchKey;
    Declare_searchKey -> Search_40;
    Search_40 -> End;
}
```

36. Implementing a Linked List with various operations:

```
class ListNode {
    int val;
    ListNode next;

    public ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class LinkedList {
    private ListNode head;

    public LinkedList() {
        this.head = null;
    }

    public void insertAtBeginning(int val) {
        ListNode newNode = new ListNode(val);
        newNode.next = this.head;
        this.head = newNode;
    }

    public void insertAtEnd(int val) {
        ListNode newNode = new ListNode(val);
        if (this.head == null) {
            this.head = newNode;
            return;
        }
        ListNode current = this.head;
        while (current.next != null) {
            current = current.next;
        }
        current.next = newNode;
    }

    public void delete(int val) {
        if (this.head == null) {
            return;
        }
        if (this.head.val == val) {
            this.head = this.head.next;
            return;
        }
        ListNode current = this.head;
        while (current.next != null && current.next.val != val) {
            current = current.next;
        }
        if (current.next != null) {
            current.next = current.next.next;
        }
    }

    public void display() {
        ListNode current = this.head;
        while (current != null) {
            System.out.print(current.val + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.insertAtEnd(10);
        list.insertAtEnd(20);
        list.insertAtBeginning(5);
        list.insertAtEnd(30);
        list.display();
        list.delete(20);
        list.display();
    }
}
```

Properly formatted dot string for the Above Implementing a Linked List with various operations is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_list [label="LinkedList list = new LinkedList();"];
    Insert_end [label="list.insertAtEnd(10);\nlist.insertAtEnd(20);\nlist.insertAtEnd(30);"];
    Insert_beginning [label="list.insertAtBeginning(5);"];
    Display_initial [label="list.display();"];
    Delete_element [label="list.delete(20);"];
    Display_final [label="list.display();"];
    End [label="End"];

    Start -> Declare_list;
    Declare_list -> Insert_end;
    Insert_end -> Insert_beginning;
    Insert_beginning -> Display_initial;
    Display_initial -> Delete_element;
    Delete_element -> Display_final;
    Display_final -> End;
}
```

37. Implementing a simple calculator using Object-Oriented Programming (OOP) principles:

```
import java.util.Scanner;

interface Operation {
    double perform(double operand1, double operand2);
}

class Add implements Operation {
    public double perform(double operand1, double operand2) {
        return operand1 + operand2;
    }
}

class Subtract implements Operation {
    public double perform(double operand1, double operand2) {
        return operand1 - operand2;
    }
}

class Multiply implements Operation {
    public double perform(double operand1, double operand2) {
        return operand1 * operand2;
    }
}

class Divide implements Operation {
    public double perform(double operand1, double operand2) {
        if (operand2 == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return operand1 / operand2;
    }
}
```

```
public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = scanner.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();
        System.out.print("Enter operator (+, -, *, /): ");
        char operator = scanner.next().charAt(0);
        double result;
        Operation operation;
        switch (operator) {
            case '+':
                operation = new Add();
                break;
            case '-':
                operation = new Subtract();
                break;
            case '*':
                operation = new Multiply();
                break;
            case '/':
                operation = new Divide();
                break;
            default:
                System.out.println("Invalid operator");
                return;
        }
        result = operation.perform(num1, num2);
        System.out.println("Result: " + result);
    }
}
```

Properly formatted dot string for the Above Implementing a simple calculator using Object-Oriented Programming (OOP) principles is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Declare_scanner [label="Scanner scanner = new Scanner(System.in);"];
    Input_num1 [label="Enter first number:"];
    Read_num1 [label="double num1 = scanner.nextDouble();"];
    Input_num2 [label="Enter second number:"];
    Read_num2 [label="double num2 = scanner.nextDouble();"];
    Input_operator [label="Enter operator (+, -, *, /):"];
    Read_operator [label="char operator = scanner.next().charAt(0);"];
    Declare_result [label="double result;"];
    Declare_operation [label="Operation operation;"];
    Switch_operator [label="switch (operator)"];
    Case_add [label="case '+':\noperation = new Add();"];
    Case_subtract [label="case '-':\noperation = new Subtract();"];
    Case_multiply [label="case '*':\noperation = new Multiply();"];
    Case_divide [label="case '/':\noperation = new Divide();"];
    Default_case [label="default:\nSystem.out.println(\"Invalid operator\");\nreturn;"];
    Perform_operation [label="result = operation.perform(num1, num2);"];
    Print_result [label="System.out.println(\"Result: \" + result);"];
    End [label="End"];

    Start -> Declare_scanner;
    Declare_scanner -> Input_num1;
    Input_num1 -> Read_num1;
    Read_num1 -> Input_num2;
    Input_num2 -> Read_num2;
    Read_num2 -> Input_operator;
    Input_operator -> Read_operator;
    Read_operator -> Declare_result;
    Declare_result -> Declare_operation;
    Declare_operation -> Switch_operator;
    Switch_operator -> Case_add [label="+"];
    Switch_operator -> Case_subtract [label="-"];
    Switch_operator -> Case_multiply [label="*"];
    Switch_operator -> Case_divide [label="/"];
    Switch_operator -> Default_case [label="default"];
    Case_add -> Perform_operation -> Print_result;
    Case_subtract -> Perform_operation -> Print_result;
    Case_multiply -> Perform_operation -> Print_result;
    Case_divide -> Perform_operation -> Print_result;
    Default_case -> End;
}
```

38. Matrix Multiplication:

```
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] matrix1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int[][] matrix2 = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
        int[][] result = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        System.out.println("Resultant Matrix after Multiplication:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Properly formatted dot string for the Above Matrix Multiplication is:

```
digraph ControlFlowDiagram {
  node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
  Start [label="Start"];
  Declare_matrices [label="Declare matrices and initialize"];
  Initialize_result [label="Initialize result matrix"];
  Loop_i [label="for (int i = 0; i < 3; i++)"];
  Loop_j [label="for (int j = 0; j < 3; j++)"];
  Loop_k [label="for (int k = 0; k < 3; k++)"];
  Multiply [label="result[i][j] += matrix1[i][k] * matrix2[k][j];"];
  Print_result [label="Print resultant matrix"];
  End [label="End"];

  Start -> Declare_matrices;
  Declare_matrices -> Initialize_result;
  Initialize_result -> Loop_i;
  Loop_i -> Loop_j [label="true"];
  Loop_j -> Loop_k [label="true"];
  Loop_k -> Multiply [label="true"];
  Multiply -> Loop_k [label="true"];
  Loop_k -> Loop_j [label="false"];
  Loop_j -> Loop_i [label="false"];
  Loop_i -> Print_result [label="false"];
  Print_result -> End;
}
```

39. Fibonacci Series with Dynamic Programming:

```
public class FibonacciDynamic {
  public static void main(String[] args) {
    int n = 10;
    int[] fibonacci = new int[n];
    fibonacci[0] = 0;
    fibonacci[1] = 1;

    for (int i = 2; i < n; i++) {
      fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
    }

    System.out.println("Fibonacci Series:");
    for (int i = 0; i < n; i++) {
      System.out.print(fibonacci[i] + " ");
    }
  }
}
```

Properly formatted dot string for the Above Fibonacci Series with Dynamic Programming is:

```
digraph ControlFlowDiagram {
  node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
  Start [label="Start"];
  Initialize_n [label="Initialize n"];
  Declare_array [label="Declare and initialize Fibonacci array"];
  Loop_i [label="for (int i = 2; i < n; i++)"];
  Calculate_fibonacci [label="fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];"];
```

```
    Print_fibonacci [label="Print Fibonacci series"];
    End [label="End"];

    Start -> Initialize_n;
    Initialize_n -> Declare_array;
    Declare_array -> Loop_i;
    Loop_i -> Calculate_fibonacci [label="true"];
    Calculate_fibonacci -> Loop_i;
    Loop_i -> Print_fibonacci [label="false"];
    Print_fibonacci -> End;
}
```

40. Printing Pascal's Triangle:

```
public class PascalTriangle {
    public static void main(String[] args) {
        int rows = 5;
        for (int i = 0; i < rows; i++) {
            int number = 1;
            for (int j = 0; j <= i; j++) {
                System.out.print(number + " ");
                number = number * (i - j) / (j + 1);
            }
            System.out.println();
        }
    }
}
```

Properly formatted dot string for the Above Printing Pascal's Triangle is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_rows [label="Initialize rows"];
    Loop_i [label="for (int i = 0; i < rows; i++)"];
    Initialize_number [label="Initialize number to 1"];
    Loop_j [label="for (int j = 0; j <= i; j++)"];
    Print_number [label="Print number"];
    Calculate_next_number [label="Calculate next number"];
    Update_number [label="Update number"];

    Start -> Initialize_rows;
    Initialize_rows -> Loop_i;
    Loop_i -> Initialize_number [label="true"];
    Initialize_number -> Loop_j;
    Loop_j -> Print_number [label="true"];
    Print_number -> Calculate_next_number;
    Calculate_next_number -> Update_number;
    Update_number -> Loop_j [label="true"];
    Loop_j -> Loop_i [label="false"];
    Loop_i -> End [label="false"];
    End [label="End"];
}
```

41. Printing a Diamond Pattern:

```
public class DiamondPattern {
    public static void main(String[] args) {
        int n = 5;
        int spaces = n - 1;
        int stars = 1;

        for (int i = 1; i <= n * 2 - 1; i++) {
            for (int j = 1; j <= spaces; j++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= stars; j++) {
                System.out.print("*");
            }
            System.out.println();

            if (i < n) {
                spaces--;
                stars += 2;
            } else {
                spaces++;
                stars -= 2;
            }
        }
    }
}
```

Properly formatted dot string for the Above Printing a Diamond Pattern is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables: n, spaces, stars"];
    Loop_i [label="for (int i = 1; i <= n * 2 - 1; i++)"];
    Loop_spaces [label="for (int j = 1; j <= spaces; j++)"];
    Print_space [label="Print space"];
    Loop_stars [label="for (int j = 1; j <= stars; j++)"];
    Print_star [label="Print star"];
    Update_spaces_stars [label="Update spaces and stars"];
    Check_condition [label="if (i < n)"];
    Update_spaces_stars_else [label="Update spaces and stars"];

    Start -> Initialize_variables;
    Initialize_variables -> Loop_i;
    Loop_i -> Loop_spaces [label="true"];
    Loop_spaces -> Print_space;
    Print_space -> Loop_spaces [label="true"];
    Loop_spaces -> Loop_stars [label="false"];
    Loop_stars -> Print_star;
    Print_star -> Loop_stars [label="true"];
    Loop_stars -> Update_spaces_stars [label="false"];
    Update_spaces_stars -> Check_condition;
    Check_condition -> Update_spaces_stars_else [label="true"];
    Update_spaces_stars_else -> Loop_i;
    Loop_i -> End [label="false"];
    End [label="End"];
}
```

42. Printing Prime Numbers up to a Given Number:

```
public class PrimeNumbers {
    public static void main(String[] args) {
        int n = 50;

        for (int i = 2; i <= n; i++) {
            boolean isPrime = true;
            for (int j = 2; j <= Math.sqrt(i); j++) {
                if (i % j == 0) {
                    isPrime = false;
                    break;
                }
            }
            if (isPrime) {
                System.out.print(i + " ");
            }
        }
    }
}
```

Properly formatted dot string for the Above Printing Prime Numbers up to a Given Number is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_n [label="Initialize n"];
    Loop_i [label="for (int i = 2; i <= n; i++)"];
    Initialize_isPrime [label="Initialize isPrime"];
    Loop_j [label="for (int j = 2; j <= Math.sqrt(i); j++)"];
    Check_prime [label="if (i % j == 0)"];
    Update_isPrime [label="Update isPrime"];
    Print_prime [label="Print prime number"];

    Start -> Initialize_n;
    Initialize_n -> Loop_i;
    Loop_i -> Initialize_isPrime;
    Initialize_isPrime -> Loop_j;
    Loop_j -> Check_prime [label="true"];
    Check_prime -> Update_isPrime [label="true"];
    Update_isPrime -> Loop_j;
    Check_prime -> Loop_j [label="false"];
    Loop_j -> Loop_i [label="false"];
    Loop_i -> Print_prime [label="true"];
    Print_prime -> Loop_i;
    Loop_i -> End [label="false"];
    End [label="End"];
}
```

43. Printing the Factorial of a Number Using a Loop:

```
public class FactorialLoop {
    public static void main(String[] args) {
        int n = 5;
        int factorial = 1;

        for (int i = 1; i <= n; i++) {
            factorial *= i;
        }

        System.out.println("Factorial of " + n + " = " + factorial);
    }
}
```

Properly formatted dot string for the Above Printing the Factorial of a Number Using a Loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_n [label="Initialize n"];
    Initialize_factorial [label="Initialize factorial"];
    Loop [label="for (int i = 1; i <= n; i++)"];
    Update_factorial [label="Update factorial"];
    Print_result [label="Print factorial"];
    End [label="End"];

    Start -> Initialize_n;
    Initialize_n -> Initialize_factorial;
    Initialize_factorial -> Loop;
    Loop -> Update_factorial [label="true"];
    Update_factorial -> Loop;
    Loop -> Print_result [label="false"];
    Print_result -> End;
}
```

44. Counting the Number of Words in a Sentence Using a Loop:

```
public class WordCountLoop {
    public static void main(String[] args) {
        String sentence = "Java programming is interesting";
        int count = 1;

        for (int i = 0; i < sentence.length(); i++) {
            if (sentence.charAt(i) == ' ') {
                count++;
            }
        }

        System.out.println("Number of words: " + count);
    }
}
```

Properly formatted dot string for the Above Counting the Number of Words in a Sentence Using a Loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_sentence [label="Initialize sentence"];
    Initialize_count [label="Initialize count"];
    Loop [label="for (int i = 0; i < sentence.length(); i++)"];
    Check_character [label="if (sentence.charAt(i) == ' ')"];
    Increment_count [label="Increment count"];
    End [label="End"];

    Start -> Initialize_sentence;
    Initialize_sentence -> Initialize_count;
    Initialize_count -> Loop;
    Loop -> Check_character [label="true"];
    Check_character -> Increment_count [label="true"];
    Increment_count -> Loop;
    Loop -> End [label="false"];
}
```

45. Finding the GCD of Two Numbers Using Euclid's Algorithm:

```
public class GCDAlgorithm {
    public static void main(String[] args) {
        int num1 = 48, num2 = 18;

        while (num2 != 0) {
            int temp = num2;
            num2 = num1 % num2;
            num1 = temp;
        }

        System.out.println("GCD: " + num1);
    }
}
```

Properly formatted dot string for the Above Finding the GCD of Two Numbers Using Euclid's Algorithm is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables"];
    Loop [label="while (num2 != 0)"];
    Swap_variables [label="Swap variables"];
    Update_num2 [label="Update num2"];
    Update_num1 [label="Update num1"];
    End [label="End"];

    Start -> Initialize_variables;
    Initialize_variables -> Loop;
    Loop -> Swap_variables [label="true"];
    Swap_variables -> Update_num2;
    Update_num2 -> Loop;
    Loop -> Update_num1 [label="false"];
    Update_num1 -> End;
}
```

46. Calculating the Sum of Natural Numbers Using a Loop:

```
public class SumOfNaturalNumbers {
    public static void main(String[] args) {
        int n = 10;
        int sum = 0;

        for (int i = 1; i <= n; i++) {
            sum += i;
        }

        System.out.println("Sum of first " + n + " natural numbers: " + sum);
    }
}
```

Properly formatted dot string for the Above Calculating the Sum of Natural Numbers Using a Loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables"];
    Loop [label="for (int i = 1; i <= n; i++)"];
    Update_sum [label="Update sum"];
    End [label="End"];

    Start -> Initialize_variables;
    Initialize_variables -> Loop;
    Loop -> Update_sum [label="true"];
    Update_sum -> Loop;
    Loop -> End [label="false"];
}
```

47. Nested loop:

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

    int weeks = 3;
    int days = 7;

    // outer loop prints weeks
    for (int i = 1; i <= weeks; ++i) {
      System.out.println("Week: " + i);

      // inner loop prints days
      for (int j = 1; j <= days; ++j) {
        System.out.println("  Day: " + j);
      }
    }
  }
}
```

Properly formatted dot string for the Above Nested loop is:

```
digraph ControlFlowDiagram {
    node [shape=rectangle, style=filled, fillcolor=lightblue, fontname=Arial];
    Start [label="Start"];
    Initialize_variables [label="Initialize variables"];
    Outer_Loop [label="for (int i = 1; i <= weeks; ++i)"];
    Print_Week [label="Print week"];
    Inner_Loop [label="for (int j = 1; j <= days; ++j)"];
    Print_Day [label="Print day"];
    End [label="End"];

    Start -> Initialize_variables;
    Initialize_variables -> Outer_Loop;
    Outer_Loop -> Print_Week [label="true"];
    Print_Week -> Inner_Loop;
    Inner_Loop -> Print_Day [label="true"];
    Print_Day -> Inner_Loop;
    Inner_Loop -> Outer_Loop [label="false"];
    Outer_Loop -> End [label="false"];
}
```