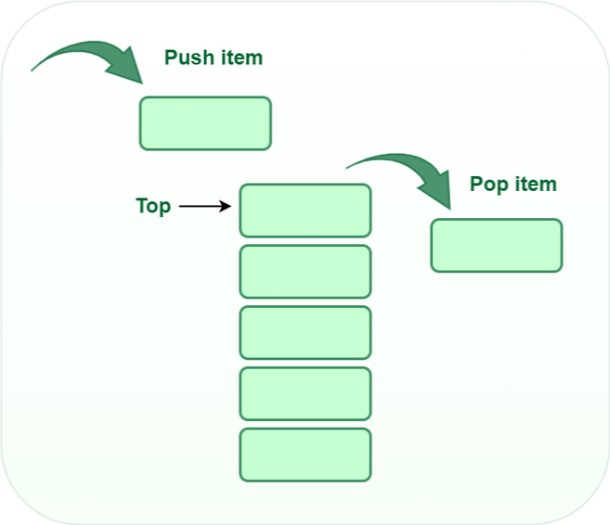
1. a. for the code following is the output: -

1 4 17 73 337

b. Big-Oh(O) may be i as it is the multiplication factor

c. Representation may be like this, (if you get another solution, please send it me also )



1. a.

import java.util.Scanner;

public class Q2  {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the String: ");

        String str = sc.next();

        System.out.println("All the characters are distinct: "+hasUniqueChars(str));

        sc.close();

    }

    public static boolean hasUniqueChars(String str) {

        for (int i = 0; i < str.length(); i++) {

            for (int j = i + 1; j < str.length(); j++) {

                if (str.charAt(i) == str.charAt(j)) {

                    return false;

                }

            }

        }

        return true;

    }

}

Output:- If a string has a character more than one time it will be false else it will be true .

b.

import java.util.Scanner;

class Student{

    String name;

    int mark;

    public Student(String name, int mark){

        this.name=name;

        this.mark=mark;

    }

}

class MarksOutOfBoundException extends Exception{

    public MarksOutOfBoundException(String message) {

        super(message);

    }

}

public class Q3 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the name of the Student: ");

        String name=sc.next();

        System.out.print("Enter mark: ");

        int mark=sc.nextInt();

        try {

            if(mark>100) {

                throw new MarksOutOfBoundException("Marks can't be greater than 100 ");

            }

            Student student=new Student(name, mark);

            System.out.print(student.name+" has got "+ student.mark);

        }

        catch(MarksOutOfBoundException e) {

            System.out.print(e);

        }

        finally {

            sc.close();

        }

    }

}

Output:-

Enter the name of the Student: Sam

Enter mark: 102

MarksOutOfBoundException: Marks can't be greater than 100

c. The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

1. a.

import java.util.Scanner;

class Student {

    int regdno;

    String name;

    double cgpa;

    Student(int regdno, String name, double cgpa) {

        this.regdno = regdno;

        this.name = name;

        this.cgpa = cgpa;

    }

    public int getRegdno() {

        return regdno;

    }

    public String getName() {

        return name;

    }

    public double getCgpa() {

        return cgpa;

    }

}

b.

class Student1 {

    int regdno;

    String name;

    double cgpa;

    Student1(int regdno, String name, double cgpa) {

        this.regdno = regdno;

        this.name = name;

        this.cgpa = cgpa;

    }

    public int getRegdno() {

        return regdno;

    }

    public String getName() {

        return name;

    }

    public double getCgpa() {

        return cgpa;

    }

}

public class Q3b {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter registration number: ");

        int regdno = sc.nextInt();

        System.out.print("Enter name: ");

        String name = sc.next();

        System.out.print("Enter CGPA: ");

        double cgpa = sc.nextDouble();

        Student1 student = new Student1(regdno, name, cgpa);

        System.out.println(student.getRegdno() + " " + student.getName() + " " + student.getCgpa());

        sc.close();

    }

}

Output- you can get yourself otherwise, you can ask me

c.

import java.util.Scanner;

class Complex {

    int real, imag;

    public void setData(int r, int i) {

        real = r;

        imag = i;

    }

    public void display() {

        System.out.println(real + " + " + imag + "i");

    }

    public Complex add(Complex c) {

        Complex temp = new Complex();

        temp.real = real + c.real;

        temp.imag = imag + c.imag;

        return temp;

    }

}

public class Q2 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Complex c1 = new Complex();

        Complex c2 = new Complex();

        System.out.print("Enter real and imaginary parts of first complex number:");

        int r1 = sc.nextInt(), i1 = sc.nextInt();

        System.out.print("Enter real and imaginary parts of second complex number:");

        int r2 = sc.nextInt(), i2 = sc.nextInt();

        c1.setData(r1, i1);

        c2.setData(r2, i2);

        sc.close();

        Complex sum = c1.add(c2);

        System.out.print("Sum of two complex numbers is: ");

        sum.display();

    }

}

Output:-

Enter real and imaginary parts of first complex number:5 10

Enter real and imaginary parts of second complex number:5 10

Sum of two complex numbers is: 10 + 20i

1. a.

abstract class Shape {

    abstract void area();

}

class Square extends Shape {

    int side;

    Square(int side) {

        this.side = side;

    }

    void area() {

        System.out.println("Area of square: " + (side \* side));

    }

}

class Triangle extends Shape {

    int base;

    int height;

    Triangle(int base, int height) {

        this.base = base;

        this.height = height;

    }

    void area() {

        System.out.println("Area of triangle: " + (0.5 \* base \* height));

    }

}

class Circle extends Shape {

    int radius;

    Circle(int radius) {

        this.radius = radius;

    }

    void area() {

        System.out.println("Area of circle: " + (3.14 \* radius \* radius));

    }

}

public class Q6 {

    public static void main(String[] args) {

        Square square = new Square(5);

        square.area();

        Triangle triangle = new Triangle(5, 10);

        triangle.area();

        Circle circle = new Circle(5);

        circle.area();

    }

}

Output- you can get yourself otherwise, you can ask me

b.

import java.util.Scanner;

public class Q4b {

    public static int gcd(int m, int n) {

        if (n == 0) {

            return m;

        } else {

            return gcd(n, m % n);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the first integer: ");

        int m = sc.nextInt();

        System.out.print("Enter the second integer: ");

        int n = sc.nextInt();

        int result = gcd(m, n);

        System.out.println("The GCD of " + m + " and " + n + " is " + result);

        sc.close();

    }

}

Output-

Enter the first integer: 45

Enter the second integer: 5

The GCD of 45 and 5 is 5

c.

import java.util.Scanner;

public class Q4c {

    public static void reverse(int n) {

        if (n < 10) {

            System.out.print(n);

        } else {

            System.out.print(n % 10);

            reverse(n / 10);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter an integer: ");

        int n = sc.nextInt();

        System.out.print("The integer with its digits reversed is: ");

        reverse(n);

        sc.close();

    }

}

Output-

Enter an integer: 4561

The integer with its digits reversed is: 1654

1. a.

Postfix expression- P Q R \* D / - E M ^ +

b.

To convert the infix expression `P − Q ∗ R/D + EˆM` into its equivalent postfix expression using stack representation, we need to perform the following PUSH and POP operations:

| Symbol | Operation |
| --- | --- |
| P | PUSH |
| Q | PUSH |
| R | PUSH |
| \* | POP |
| Q | POP |
| R | POP |
| \* | PUSH |
| D | PUSH |
| / | POP |
| P | POP |
| Q | POP |
| R | POP |
| \* | PUSH |
| D | PUSH |
| / | PUSH |
| - | PUSH |
| E | PUSH |
| M | PUSH |
| ^ | POP |
| E | POP |
| M | POP |
| ^ | PUSH |
| + | PUSH |

Therefore, there are 10 PUSH operations and 9 POP operations during the conversion of the infix expression `P − Q ∗ R/D + EˆM` into its equivalent postfix expression using stack representation.

c.

| Symbol | Operation |
| --- | --- |
| 5 | PUSH |
| 2 | PUSH |
| 3 | PUSH |
| - | POP |
| 2 | POP |
| 3 | POP |
| - | PUSH |
| 25 | PUSH |
| 5 | PUSH |
| / | POP |
| 25 | POP |
| 5 | POP |
| / | PUSH |
| ∗ | POP |
| -1 | POP |
| ∗ | PUSH |

To evaluate the postfix expression `5, 2, 3, −, 25, 5, /, ∗, +` using stack representation, we need to perform the following PUSH and POP operations:

Therefore, there are 9 PUSH operations and 8 POP operations during the evaluation of the postfix expression `5, 2, 3, −, 25, 5, /, ∗, +` using stack representation.

1. a.

import java.util.LinkedList;

import java.util.Queue;

public class Q6a {

    public static void main(String[] args) {

        Queue<Integer> queue = new LinkedList<>();

        queue.add(10);

        queue.add(20);

        queue.add(30);

        queue.add(40);

        System.out.println("Queue before deletion: " + queue);

        queue.remove();

        System.out.println("Queue after deletion: " + queue);

    }

}

Output-

Queue before deletion: [10, 20, 30, 40]

Queue after deletion: [20, 30, 40]

b.

| Operation | Top | Contents |
| --- | --- | --- |
| push(‘A’) | A | A |
| push(‘T’) | T | T, A |
| push(‘K’) | K | K, T, A |
| push(‘M’) | M | M, K, T |
| pop() | K | K, T |
| pop() | T | T |
| push(‘D’) | D | D, T |
| push(‘W’) | W | W, D, T |
| push(‘S’) | S | S, W, D |

c.

| **Stacks** | **Queues** |
| --- | --- |
| A stack is a data structure that stores a collection of elements, with operations to push (add) and pop (remove) elements from the top of the stack. | A queue is a data structure that stores a collection of elements, with operations to enqueue (add) elements at the back of the queue, and dequeue (remove) elements from the front of the queue. |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Stacks are often used for tasks that require backtracking, such as parsing expressions or implementing undo functionality. | Queues are often used for tasks that involve processing elements in a specific order, such as handling requests or scheduling tasks. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Stacks are implemented using an array or linked list data structure. | Queues are implemented using an array or linked list data structure. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on [recursion](https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/). | Queue is used in solving problems having sequential processing. |
| Stacks are often used for recursive algorithms or for maintaining a history of function calls. | Queues are often used in multithreaded applications, where tasks are added to a queue and executed by a pool of worker threads. |
| Stack does not have any types. | Queue is of three types – 1. Circular Queue 2. Priority queue 3. double-ended queue. |
| Can be considered as a vertical collection visual. | Can be considered as a horizontal collection visual. |
| Examples of stack-based languages include PostScript and Forth. | Examples of queue-based algorithms include Breadth-First Search (BFS) and printing a binary tree level-by-level. |

Any 4 or 5 you can do.

1. a.

class Node {

    int data;

    Node next;

    public Node(int data) {

        this.data = data;

        this.next = null;

    }

}

public class Q7a {

    public static void main(String[] args) {

        // Create a linked list

        Node head = new Node(87);

        head.next = new Node(23);

        head.next.next = new Node(34);

        head.next.next.next = new Node(66);

        // Create a new node with data 45

        Node newNode = new Node(45);

        // Find the node with data 23

        Node currentNode = head;

        while (currentNode.data != 23) {

            currentNode = currentNode.next;

        }

        // Insert the new node after the node with data 23

        newNode.next = currentNode.next;

        currentNode.next = newNode;

        // Print the linked list

        printLinkedList(head);

    }

    private static void printLinkedList(Node head) {

        Node currentNode = head;

        while (currentNode != null) {

            System.out.println(currentNode.data);

            currentNode = currentNode.next;

        }

    }

}

Output –

87

23

45

34

66

b.

public class Q7b {

    Node head;

    static class Node {

        int data;

        Node next;

        Node(int d) {

            data = d;

            next = null;

        }

    }

    public void insertAtEnd(int data) {

        Node newNode = new Node(data);

        if (head == null) {

            head = newNode;

            return;

        }

        Node current = head;

        while (current.next != null) {

            current = current.next;

        }

        current.next = newNode;

    }

    public void printList() {

        Node current = head;

        while (current != null) {

            System.out.print(current.data + " ");

            current = current.next;

        }

    }

    public static void main(String[] args) {

        LinkedList list = new LinkedList();

        list.insertAtEnd(1);

        list.insertAtEnd(2);

        list.insertAtEnd(3);

        list.printList();

    }

}

Output- 1 2 3

c.

Multiple inheritances lead to ambiguity.

For example, if there is a class named Sub and there are two classes Super1 and Super2 and if both contains a method named **sample()**.

 And if the class sub inherits both classes Super1 and Super2 then there will be two copies of the sampling method one from each superclass and it is ambiguous to decide which method to be executed.

1. a.

int count = 0;

Node current = p1;

while (current != null) {

count++;

current = current.next;

}

System.out.println("Total number of nodes: " + count);

b.

if (p1 == null) {

return;

}

if (p1.next != null) {

p1.data = p1.next.data;

p1.next = p1.next.next;

} else {

Node current = head;

while (current.next != p1) {

current = current.next;

}

current.next = null;

}

c.

public int sumOfOddElements(Node head) {

int sum = 0;

Node current = head;

while (current != null) {

if (current.data % 2 != 0) {

sum += current.data;

}

current = current.next;

}

return sum;

}

1. a.

An almost complete binary tree can be constructed with 10 nodes, but a complete binary tree cannot.

The maximum number of nodes in a complete binary tree with height h is 2h + 1 - 1. The height of a complete binary tree with 10 nodes would be log2(10 + 1) = 3.105. Since the maximum number of nodes in a complete binary tree with height 3 is 2 \* 3 + 1 - 1 = 7, a complete binary tree with 10 nodes cannot be constructed.

However, an almost complete binary tree can be constructed with 10 nodes. An almost complete binary tree is a binary tree where all levels except the last level are completely filled, and the last level is filled from left to right as much as possible. In a binary tree with 10 nodes, the last level can have at most 2 nodes, so the last level can be filled completely. Therefore, an almost complete binary tree can be constructed with 10 nodes.

Here is an example of an almost complete binary tree with 10 nodes:

Code snippet

1

/ \

2 3

/ \ / \

4 5 6

**OR**

A complete binary tree is a binary tree type where all the leaf nodes must be on the same level. However, root and internal nodes in a complete binary tree can either have 0, 1 or 2 child nodes1.

An almost complete binary tree is the binary tree made up of the first n nodes of a canonically labeled complete Binary Tree2.

Since you have 10 nodes, you can construct an almost complete binary tree among almost complete binary tree and complete binary tree2.

b. Try it yourself or ask if needed

c. In order Traversal: - S, T, K, Z, H, F, C, A, L, M, R

1. a.

20

/ \

/ \

/ \

/ \

12 21

/ \ \

/ \ \

/ \ \

5 14 18

/ \ / \

/ \ / \

/ \ / \

13 16 19 17

\

\

9

\

\

7

b.

To represent a binary tree as an array, we use a technique called level-order traversal. In level-order traversal, the elements of the binary tree are stored in the array in the order in which they are visited in a breadth-first search¹.

Here is the array representation of the binary search tree given in fig.1:

```

[20, 12, 21, 5, 14, 18, 13, 19, 16, 17, 9, 7]

```