PRESIDENCY UNIVERSITY

MODULE 1

Local variables

The variables declared inside a method or a block are known as local variables. A local variable is visible within the method in which it is declared. The local variable is created when execution control enters into the method or block and is destroyed after the method or block execution is completed.

```
/* Java program to demonstrate Local variables */

public class LocalVariables {

	public void show() {
		int a = 10;
		//static int x = 100;
		System.out.println("Inside show method, a = " + a);
	}
	public void display() {
		int b = 20;
		System.out.println("Inside display method, b = " + b);
		// trying to access variable 'a' - generates an ERROR
		System.out.println("Inside display method, a = " + a);
	}
	public static void main(String args[]) {
		LocalVariables obj = new LocalVariables();
		obj.show();
		obj.display();
	}
}
```

Instance variables or member variables or global variables

The variables declared inside a class and outside any method, constructor or block are known as instance variables or member variables. These variables are visible to all the methods of the class. The changes made to these variables by method affects all the methods in the class. These variables are created separate copy for every object of that class.

```
/* Java program to demonstrate Global variables */

public class ClassVariables {

	int x = 100;

	public void show() {
		System.out.println("Inside show method, x = " + x);
		x = x + 100;
	}
	public void display() {
		System.out.println("Inside display method, x = " + x);
	}

	public static void main(String[] args) {
		ClassVariables obj = new ClassVariables();
		obj.show();
		obj.display();
	}
}
```

Static variables or Class variables

A static variable is a variable that is declared using static keyword. The instance variables can be static variables but local variables can not. Static variables are initialized only once, at the start of the program execution. The static variable only has one copy per class irrespective of how many objects we create.

```
/* Java program to demonstrate Static variables */

public class StaticVariablesExample
{
	int x, y;  // Instance variables
	static int z;  // Static variable

	StaticVariablesExample(int x, int y)
	  {
		this.x = x;
		this.y = y;
	}
	public void show() {
		    int a;  // Local variables
		System.out.println("Inside show method,");
		System.out.println("x = " + x + ", y = " + y + ", z = " + z);
	}

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

```
		StaticVariablesExample obj_1 = new StaticVariablesExample(10, 20);
		StaticVariablesExample obj_2 = new StaticVariablesExample(100, 200);
		obj_1.show();
		StaticVariablesExample.z = 1000;
		obj_2.show();
	}
}
```

Final variables

A final variable is a variable that declared using final keyword. The final variable is initialized only once, and does not allow any method to change it's value again. The variable created using final keyword acts as constant. All variables like local, instance, and static variables can be final variables.

```
/* Java program to demonstrate Final variables */

public class FinalVariableExample
{
	final int a = 10;
	void show() {
		System.out.println("a = " + a);
		a = 20; //Error due to final variable cann't be modified
	}

	public static void main(String[] args) {

		FinalVariableExample obj = new FinalVariableExample();
		obj.show();

	}

}
```

ARRAYS in Java

Creating an array

Syntax :

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

Instantiation of an Array in Java:

```
arrayRefVar=new datatype[size];
```

//Java Program to illustrate how to declare, instantiate, initialize and traverse the Java array.

```
class Testarray
{
	public static void main(String args[])
	{
			int a[]=new int[5];	//declaration and instantiation
			a[0]=10;		//initialization
			a[1]=20;
			a[2]=70;
			a[3]=40;
			a[4]=50;
	//traversing array
	for(int i=0;i<a.length;i++)		//length is the property of array
		System.out.println(a[i]);
	}
}
```

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1
{
	public static void main(String args[])
	{
	int a[]={33,3,4,5};	//declaration, instantiation and initialization
	//printing array
		for(int i=0;i<a.length;i++)	//length is the property of array
			System.out.println(a[i]);
	}
}
```

```
//Java Program to print the array elements using for-each loop

class Testarray1
{
	public static void main(String args[])
	 {
		int arr[]={33,3,4,5};
	//printing array using for-each loop
		for(int i:arr)
			System.out.println(i);
	}
}


//Java Program to demonstrate the way of passing an array to method.

public class TestArray
{

	//creating a method which receives an array as a parameter
	static void printArray(int arr[])
	{
		for(int i=0;i<arr.length;i++)
			System.out.println(arr[i]);
	}

	public static void main(String args[])
	{
		printArray(new int[]{10,22,44,66});	//passing array to method
	}
}
```

Stack Implementation using Arrays

❖ STATIC CODE:

```
package stack1;

public class Stack {
   private int maxSize;
   private int top;
   private int[] stackArray;

   public Stack() {
      maxSize = 5;
      stackArray = new int[maxSize];
      top = -1; // Empty stack
   }

   public void push(int value) {
      if (top < maxSize - 1) {
        top=top+1;
         stackArray[top] = value;
         System.out.println(value + " pushed onto the stack.");
      } else {
         System.out.println("Stack is full. Cannot push " + value);
      }
   }

   public int pop() {
      if (top >= 0) {
         int poppedValue = stackArray[top];
         top=top-1;
         System.out.println(poppedValue + " popped from the stack.");
         return poppedValue;
      } else {
         System.out.println("Stack is empty. Cannot pop.");
         return -1; // Or throw an exception
      }
   }
```

```
public int peek() {
    if (top >= 0) {
        return stackArray[top];
    } else {
        System.out.println("Stack is empty. Nothing to peek.");
        return -1; // Or throw an exception
    }
}

public boolean isEmpty() {
    return top == -1;
}

public boolean isFull() {
    return top == maxSize - 1;
}

public void display() {
      for(int i=0;i<=top;i++)
      {
              System.out.println("The elements in array are:");
              System.out.println(stackArray[i]);
      }
}

public static void main(String[] args) {
    Stack stack = new Stack();

    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.display();

    System.out.println("Peek: " + stack.peek());

    stack.pop();
    stack.pop();
    stack.push(50);
    stack.display();
```

```
        System.out.println("Is empty? " + stack.isEmpty());
        System.out.println("Is full? " + stack.isFull());

    }
}
```

[OR]

❖ Using SCANNER CLASS (DYNAMIC CODE)

```
package stack1;
import java.util.Scanner;
public class Stacknew {
    private int maxSize;
            private int top;
            private int[] stackArray;

            public Stacknew(int size) {
                maxSize = size;
                stackArray = new int[maxSize];
                top = -1;
            }

            public boolean isEmpty() {
                return top == -1;
            }

            public boolean isFull() {
                return top == maxSize - 1;
            }

            public void push(int item) {
                if (isFull()) {
                    System.out.println("Stack is full. Cannot push.");
                    return;
                }
                stackArray[++top] = item;
                System.out.println("Pushed: " + item);
            }
```

```
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty. Cannot pop.");
            return -1; // You can choose a different value to represent an error.
        }
        int item = stackArray[top--];
        System.out.println("Popped: " + item);
        return item;
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty. Cannot peek.");
            return -1; // You can choose a different value to represent an error.
        }
        return stackArray[top];
    }


//public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the stack: ");
        int size = scanner.nextInt();

        Stacknew stack = new Stacknew(size);

        while (true) {
            System.out.println("\nStack Operations:");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Peek");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter element to push: ");
                    int pushItem = scanner.nextInt();
```

```
                stack.push(pushItem);
                break;
            case 2:
                int popItem = stack.pop();
                if (popItem != -1) {
                    System.out.println("Popped item: " + popItem);
                }
                break;
            case 3:
                int peekItem = stack.peek();
                if (peekItem != -1) {
                    System.out.println("Peeked item: " + peekItem);
                }
                break;
            case 4:
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
```

Evaluation of Postfix Expression:

```
package stack1;
import java.util.Stack;
import java.util.Scanner;

public class postfixEvaluation {

                public static int evaluatePostfix(String expression) {
              Stack<Integer> stack = new Stack<>();

              for (char c : expression.toCharArray()) {
                 if (Character.isDigit(c)) {
                    // If the character is a digit, push it onto the stack
                    stack.push(c - '0'); // Convert char to int and push
                 } else {
                    // If the character is an operator, pop two operands from the
stack,
                    // apply the operator, and push the result back onto the stack
                    int operand2 = stack.pop();
                    int operand1 = stack.pop();
                    int result = applyOperator(operand1, operand2, c);
                    stack.push(result);
                 }
              }

              // The final result should be on the top of the stack
              return stack.pop();
           }

 private static int applyOperator(int operand1, int operand2, char operator) {
              switch (operator) {
                 case '+':
                    return operand1 + operand2;
                 case '-':
                    return operand1 - operand2;
                 case '*':
                    return operand1 * operand2;
                 case '/':
```

```
                    if (operand2 == 0) {
                        throw new ArithmeticException("Division by zero");
                    }
                    return operand1 / operand2;
                default:
                    throw new IllegalArgumentException("Invalid operator: " +
operator);
            }
        }

        public static void main(String[] args) {
            Scanner s=new Scanner(System.in);
            System.out.println("Enter the Postfix Expression");
            String postfixExpression =s.nextLine(); // Example postfix
expression
            int result = evaluatePostfix(postfixExpression);
            System.out.println("Result of the postfix expression: " + result);
        }
    }
```

Infix to Postfix Conversion

```
package stack1;
import java.util.Stack;
public class InfixToPostfix {
                        // Function to check if a character is an operator
        private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/' || c=='^';
        }

        // Function to check the precedence of an operator
        private static int precedence(char operator) {
            switch (operator) {
                case '+':
                case '-':
                    return 1;
                case '*':
                case '/':
                    return 2;
                case '^': return 3;
            }
            return -1;
        }

        // Function to convert infix expression to postfix expression
        public static String infixToPostfix(String infixExpression) {
            StringBuilder postfix = new StringBuilder();
            Stack<Character> stack = new Stack<>();

            for (char c : infixExpression.toCharArray()) {
                if (Character.isLetterOrDigit(c)) {
                    postfix.append(c);
                } else if (c == '(') {
                    stack.push(c);
                } else if (c == ')') {
                    while (!stack.isEmpty() && stack.peek() != '(') {
                        postfix.append(stack.pop());
                    }
                    stack.pop(); // Pop the '('
                } else if (isOperator(c)) {
```

```
                while (!stack.isEmpty() && precedence(c) <=
precedence(stack.peek())) {
                    postfix.append(stack.pop());
                }
                stack.push(c);
            }
        }

        while (!stack.isEmpty()) {
            if (stack.peek() == '(') {
                return "Invalid infix expression"; // Unmatched '('
            }
            postfix.append(stack.pop());
        }

        return postfix.toString();
    }

    public static void main(String[] args) {
        String infixExpression = "A + B * C ^ D / E";
        String postfixExpression = infixToPostfix(infixExpression);
        System.out.println("Infix Expression: " + infixExpression);
        System.out.println("Postfix Expression: " + postfixExpression);
    }
}
```

Implementation of Queue using Arrays

❖ STATIC CODE:

```
package queue;

public class Uqueue {
        private int maxSize;
    private int front;
    private int rear;
    private int[] queueArray;

    public Uqueue() {
        maxSize = 5; // One extra space to differentiate between front and rear
positions
        queueArray = new int[maxSize];
        front = 0;
        rear = -1;
    }

    public void enqueue(int value) {
        if (rear==maxSize-1) {
          System.out.println("Queue is full. Cannot enqueue " + value);
          return;
          }
        else {
            rear = (rear + 1);
            queueArray[rear] = value;
            System.out.println(value + " enqueued.");
        }

        }

public int dequeue() {
    if (rear == front-1) {
      System.out.println("Queue is empty. Cannot dequeue.");
```

```
      return -1;
    } else {
      int i = front;
      int t=rear;
      int dequeuedValue = queueArray[i];
      int d = dequeuedValue;
      for (i = front; i <= t; i++) {
        System.out.println("Front is .: "+front+"  Rear is : "+rear);
        System.out.println("ith pos : "+queueArray[i]+"i+1th pos :
"+queueArray[i+1]);
        queueArray[i] = queueArray[i + 1];
        t--;
            }
      rear=rear-1;
      System.out.println("Rear after loop is"+rear);
      System.out.println(d + " dequeued.");
      return d;
    }
  }


    public int peek() {
        if (front<=rear) {
            return queueArray[front];
        } else {
            System.out.println("Queue is empty. Nothing to peek.");
            return -1; // Or throw an exception
        }
    }

    public void display1() {
         int i;
         if(isEmpty()) {
                System.out.println("Empty Queue");
         }
         else {
                System.out.println("Items in queue");
```

```
            for(i=front;i<=rear;i++) {
                System.out.println(queueArray[i]);
            }
        }
    }

    public boolean isEmpty() {
        return (rear + 1)== front;
    }

    public boolean isFull() {
        return rear== maxSize-1;
    }

    public static void main(String[] args) {
        Uqueue queue = new Uqueue();

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.display1();

        System.out.println("Peek: " + queue.peek());

        queue.dequeue();
        queue.dequeue();
        queue.dequeue();
        queue.enqueue(40);
        queue.display1();

        System.out.println("Is empty? " + queue.isEmpty());
        System.out.println("Is full? " + queue.isFull());
    }


}
```

❖ **USING SCANNER CLASS(DYNAMIC CODE):**

```
public class Queue {
  private int maxSize;
  private int front;
  private int rear;
  private int[] queueArray;

  public Queue(int size) {
    maxSize = size; // One extra space to differentiate between front and rear
positions
    queueArray = new int[maxSize];
    front = 0;
    rear = -1;
  }

  public void enqueue(int value) {
    if (rear == maxSize - 1) {
      System.out.println("Queue is full. Cannot enqueue " + value);
      return;
    } else {
      rear = (rear + 1);
      queueArray[rear] = value;
      System.out.println(value + " enqueued.");
    }

  }

  public int dequeue() {
    if (rear == front-1) {
      System.out.println("Queue is empty. Cannot dequeue.");
      return -1;
    } else {
      int i = front;
      int t=rear;
      int dequeuedValue = queueArray[i];
      int d = dequeuedValue;
```

```
      for (i = front; i <= t; i++) {
        System.out.println("Front is .: "+front+"  Rear is : "+rear);
        System.out.println("ith pos : "+queueArray[i]+"i+1th pos :
"+queueArray[i+1]);
        queueArray[i] = queueArray[i + 1];
        t--;
            }
      rear=rear-1;
      System.out.println("Rear after loop is"+rear);
      System.out.println(d + " dequeued.");
      return d;
    }
  }

  public int peek() {
    if (front <= rear) {
      System.out.println("Peek:" + queueArray[front]);
      return queueArray[front];
    } else {
      System.out.println("Queue is empty. Nothing to peek.");
      return -1; // Or throw an exception
    }
  }

  public void display() {
    int i;
    if (isEmpty()) {
      System.out.println("Empty Queue");
    } else {
      System.out.println("Items in Queue");
      for (i = front; i <= rear; i++) {
        System.out.println("Front is"+front+"Rear is"+rear+"i value is"+i);
        System.out.println(queueArray[i]);
      }
    }
  }
```

```
public boolean isEmpty() {
  return rear == front - 1;
}

public boolean isFull() {
  return rear == maxSize - 1;
}

public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  System.out.print("Enter the size of the queue: ");
  int size = scanner.nextInt();
  Queue queue = new Queue(size);
  while (true) {
    System.out.println("\nQueue Operations:");
    System.out.println("1. Enqueue");
    System.out.println("2. Dequeue");
    System.out.println("3. Peek");
    System.out.println("4. Display");
    System.out.println("5. Is Empty");
    System.out.println("6. Is Full");
    System.out.println("7. Exit");
    System.out.print("Enter your choice: ");
    int choice = scanner.nextInt();
    switch (choice) {
      case 1:
        System.out.print("Enter element to insert: ");
        int insertItem = scanner.nextInt();
        queue.enqueue(insertItem);
        break;
      case 2:
        queue.dequeue();
        break;
      case 3:
        queue.peek();
        break;
      case 4:
```

```
            queue.display();
            break;
          case 5:
            System.out.println("IS EMPTY:" + queue.isEmpty());
            break;
          case 6:
            System.out.println("IS FULL:" + queue.isFull());
            break;
          case 7:
            scanner.close();
            System.exit(0);
          default:
            System.out.println("Invalid choice. Please try again.");
        }
      }
    }
}
```