Experiment 1: Getting Familiar with Eclipse:

(a) Download and Install Eclipse.

(b) Using Eclipse for Java.

Learning Outcome:
Learnt how to install and setup the java environment to run java programs. We got familiar with the eclipse application.

Experiment 2: Write a Java program to print “Hello World” to understand compilation and execution of java program.

Theory:
Java is a high-level, object-oriented programming language known for its portability and versatility. It provides a robust runtime environment that supports automatic memory management and platform independence. Java programs are compiled into bytecode, which can run on any Java Virtual Machine (JVM). Java is widely used for building web applications, mobile apps, enterprise software, and embedded systems.

Learning Outcome:
Learnt how to run a basic java program and get output on the console.

Experiment 3: Write java program demonstrating the usage of literal datatypes.

Theory:
In Java, literals represent fixed values assigned to variables. Java supports various literal data types: Integer literals, Floating-point literals, Character literals, Boolean literals, String literals, Null literal and more. Each literal data type has specific syntax rules and represents a particular value or state in Java programs.

Learning Outcome:
Learnt how to work with different datatypes in java and using various methods for implementing literal datatypes.

Experiment 4: Write a Java program demonstrating the usage of arithmetic, assignment and unary operators.

Theory:
Arithmetic operators: perform mathematical operations on numeric operands, such as addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

Assignment operators: assign values to variables, including simple assignment (=) as well as compound assignment operators like +=, -=, *=, /=, and %=.

Unary operators: operate on a single operand, such as unary plus (+), unary minus (-), increment (++), decrement (--), logical complement (!), and bitwise complement (~).

Learning Outcome:
We learnt about applying different arithmetic on numerical data in java and assignment values to different variables and got to know about the concept of unary operators.

Experiment 5: Write a Java program to generate random number up to 100 and print whether it is prime number or not.

Theory:
In Java, random number generation is facilitated by the java.util.Random class. By creating an instance of Random, developers can generate pseudorandom numbers using methods like nextInt(), nextDouble(), or nextBoolean(). These methods produce sequences of seemingly random values based on a seed value, providing a versatile way to introduce randomness into applications for tasks such as simulations, gaming, or cryptography.

Learning Outcome:
Learnt how to generate a random value in java using random class present in java.util

Experiment 6: (a) Design a Java program to generate first 10 terms of Fibonacci.
(b) Find factorial using recursion.

Theory:

Recursion in Java involves a method calling itself to solve problems by breaking them down into smaller instances. It typically consists of a base case to terminate the recursion and a recursive case to call the method with a simpler version of the problem. Recursion can be an elegant and efficient solution for tasks like tree traversal, factorial calculation, and solving mathematical problems.

Learning Outcome:
Learnt how to implement the concept of recursion in java and use it for calculating factorial of a number.

Experiment 7: Design a Java program to find the average sum of array of N numbers entered by user.

Theory:
In Java, finding the average and sum of elements in an array involves iterating through the array and accumulating the sum of elements. The average is calculated by dividing the sum by the number of elements. This process can be implemented using a loop, providing flexibility and efficiency for handling arrays of various sizes and types.

Learning Outcome:
Learnt how to traverse through an array and apply mathematical function like average on the value present in the array.

Experiment 8: Design a Java program to implement classes and objects.
(a) Using default constructor.
(b) Using parametrized constructor.
(c) Using Copy constructor.

Theory:
In Java, a constructor is a special type of method used to initialize objects of a class. It has the same name as the class and is invoked automatically when an object of the class is created using the new keyword. Constructors can be used to set initial values for instance variables, perform initialization tasks, or allocate resources. They can be overloaded to provide multiple constructors with different parameter lists, enabling flexibility in object creation.

Learning Outcome:
Learnt how to apply the concept of constructor in java and get a better understanding of the concept.

Experiment 9: Create a class and find out the area and perimeter of rectangle

Theory:
In Java, a class is a blueprint for creating objects that define the structure and behavior of those objects. It encapsulates data (fields) and methods (functions) to operate on that data. Classes promote code reusability and maintainability by allowing similar objects to be created based on a common template.

Learning Outcome:
Learnt how to apply operation in java using multiple methods with return.

Experiment 10: Create a class circle with instance variable radius and member function
(a) area (b) circumference (c) display()
Write a test application named circletest that demonstrate class circle capabilities.

Theory:
In Java, object creation involves using a class as a blueprint to instantiate objects with the new keyword. Each object occupies memory and has its own set of instance variables and methods defined by the class. Constructors are invoked during object creation to initialize the newly created object.

Learning Outcome:
Learnt how to create object of a class in java and invoke its methods to initiate different operations.

Experiment 11: Design a class that perform string operation (equal, reverse and changeCase)

Theory:
In Java, string operations like equality comparison, reversing, and changing case are commonly performed using methods provided by the String class:

Equality comparison: Use the equals() method to check if two strings have the same content.

Reversing: Reverse a string by converting it to a StringBuilder object and using its reverse() method.

Changing case: Use methods like toUpperCase() and toLowerCase() to change the case of characters in a string.

Learning Outcome:
Learnt how to implement different methods for string manipulation and applying string operations.

Experiment 12: Write a java program to implement push and pop operation of stack. Also ensure stack overflow and underflow condition are checked while performing push and pop operations

Theory:
In Java, a stack is a data structure that follows the Last In, First Out (LIFO) principle. It supports two primary operations:

Push: Adds an element to the top of the stack.

Pop: Removes and returns the top element from the stack.

Stacks can encounter underflow (when trying to pop from an empty stack) and overflow (when trying to push onto a full stack) conditions, which should be handled to prevent runtime errors and ensure proper stack management.

Learning Outcome:
Learnt how to implement the stack data structure in java and ensure overflow and underflow condition to prevent errors.

Experiment 13: (a) Write a Java program to demonstrate passing object as parameters.
(b) Write a Java program to demonstrate the difference between call by value and call by reference.

Theory:
In Java, passing objects as parameters involves passing a reference to the object's memory location, allowing methods to modify the object's state.

Call by value means passing a copy of the reference to the object, so changes made to the object's state within the method persist outside.

Call by reference means passing the actual reference to the object, enabling modifications to the object's state to affect its external state as well.

Learning Outcome:
We learnt how to pass objects as parameters in java and also learnt how to pass parameters by call by value and call by reference technique.

Experiment 14: Write a Java program to demonstrate the concept of abstract classes and interfaces.

Theory:
In Java, abstract classes are classes that cannot be instantiated and may contain abstract methods.

Interfaces are similar to abstract classes but can only contain abstract methods and constants.

Both abstract classes and interfaces serve as blueprints for other classes to implement or extend, facilitating code reuse and promoting polymorphism.

Abstract classes provide a partial implementation, while interfaces define a contract for implementing classes, promoting loose coupling and multiple inheritance through interface implementation.

Learning Outcome:
we learnt about the implementation of abstract classes and interfaces in Java

Experiment 15: Write a Java program to implement inheritance Define a class Box with the following instance variables: width, height and depth, all of type float. Create a new class BoxWeight that extends Box to include weight as an instance variable. Write an application that tests the functionalities of both these classes.

Theory:
Inheritance in object-oriented programming allows a class (subclass) to inherit fields and methods from another class (superclass), fostering code reuse and promoting hierarchical relationships between classes.

Subclasses can extend the functionality of their superclass by adding new methods or overriding existing ones, enabling customization and specialization while maintaining consistency in behavior.

Through inheritance, polymorphism is achieved, allowing objects of different classes to be treated uniformly, enhancing code flexibility and extensibility.

Learning Outcome:
We learnt about the concept of implementation and how subclass and superclass works.

Experiment 16: Implement the following Java programs to demonstrate the concept of exception handling using keywords try, catch, finally, throw and throws wherever required

Theory:
Exception handling in Java provides a mechanism to gracefully handle runtime errors and exceptional conditions that may occur during program execution.

By using try-catch blocks, developers can detect, respond to, and recover from exceptions, ensuring robustness and reliability in their applications.

Additionally, Java's exception hierarchy enables custom exception classes to be defined, facilitating precise error reporting and effective debugging.

Learning Outcome:
We learnt about the concept of exception handling and how we can use this feature to make sure the program runs smoothly by resolving exceptions at run time.

Experiment 17: Demonstrate the use of final keyword with data member, function and class

Theory:
The `final` keyword in Java denotes immutability, allowing variables to be assigned a value only once, preventing further modifications.

When applied to methods, `final` restricts subclassing, ensuring method implementations remain unchanged in subclasses.

Using `final` with classes prohibits inheritance, safeguarding the integrity of class design and preventing extension or modification of its behavior.

Learning Outcome:
we learnt the different functionalities of the ‘final’ keyword and how it is used to restrict modification to variable or inheritance of a class.

Experiment 18: Write a Java program to demonstrate the usage of following Collections:

List: Array List
Set
Map.

Theory:
Collections in Java provide a framework for storing and manipulating groups of objects, offering a wide range of data structures like lists, sets, and maps.

They offer dynamic resizing, efficient retrieval, and insertion operations, catering to diverse needs in programming from simple storage to complex data manipulation.

Collections support generic types, enabling type-safe usage and enhancing code readability and maintainability while facilitating interoperability with other Java APIs.

Learning Outcome:
We learnt about the concept of collections in java and get the idea of how to store a group of objects together in a collection to make our code more reliable.

Experiment 19: Design a program to demonstrate multi-threading using Thread Class.

Theory:
Multi-threading allows concurrent execution of multiple threads within a single process, enabling efficient utilization of CPU resources and enhancing program responsiveness.

Threads share the same memory space but execute independently, enabling tasks such as background processing, parallel computation, and responsive user interfaces.

Effective multi-threading requires careful synchronization to manage shared resources and avoid race conditions, ensuring thread safety and correct program behavior.

Learning Outcome:
We learnt the concept of multithreading and applied this on our program to see how it works.

Experiment 20: Design a program to create game 'Tic Tac Toe'

Theory:
In Tic Tac Toe, players take turns placing their marks on a 3x3 grid, aiming to form a horizontal, vertical, or diagonal line of their marks to win.

The game employs basic programming concepts such as arrays, loops, and conditional statements to manage game state, validate moves, and determine the winner.

By providing an interactive console interface, players can engage in a classic game while gaining familiarity with fundamental programming constructs.

Learning Outcome:
We get to learn how combining different basic concepts of programming helps us to create a bigger program or a game.

Experiment 21: Design a program to read a text file and after printing that on scree write the content to another text file.

Theory:
The Java program demonstrated above involves reading from an input text file and displaying its contents on the console, followed by writing the same contents to a separate output file. This showcases basic file handling in Java, utilizing classes like `BufferedReader` and `BufferedWriter` from the `java.io` package for efficient text reading and writing. Exception handling is crucial in this process to manage potential I/O errors and ensure the program's robustness in case of issues like missing files or access rights.

Learning Outcome:
We get to learn how file management and manipulation works in java.

Experiment 22: Design a program to count number of words, characters, vowels in a text file

Theory:
The Java program provided reads from a text file, utilizing `BufferedReader` to efficiently handle line-by-line reading, which aids in counting words, characters, and vowels accurately. It splits each line into words using regular expressions, counts individual characters, and identifies vowels using a helper method that checks membership in a predefined string of vowel characters. This exercise demonstrates basic text processing capabilities in Java, showcasing how to manipulate and analyze textual data for insights such as content length and composition.

Learning Outcome:
We learnt about reading data from a text file line by line and applying string functions on it to get insightful information.

Experiment 23: Design a program to create simple chat application using Socket Programming

Theory:
Socket programming is a way to enable communication between different processes, often over a network. It uses the client-server model where typically a server listens on a specific port and a client connects to it, facilitating two-way data transfer. In Java, the `java.net` package provides tools to implement sockets, allowing TCP or UDP protocol-based communications. Socket programming is fundamental in network application development, supporting various uses such as creating web servers, chat applications, and any distributed network services.

Learning Outcome:
We learnt about the concept of socket programming where we can connect different programs of server and client establishing a connection between them.

Experiment 24: Design a program to basic calculator using Applet and Event Handling.

Theory:
An applet is a Java program that runs within a web browser, providing dynamic and interactive content on web pages, often used for graphical user interfaces and multimedia applications.

Event handling in Java allows applets to respond to user actions such as mouse clicks or keyboard inputs, enabling interactivity and dynamic behavior within the applet.

By implementing event listeners and callback methods, applets can detect and process various events, facilitating user interaction and enhancing the user experience in web-based applications.

Learning Outcome:
We learnt about event handling in java and use of applet to make a web interface for our java program to access its functions.

Experiment 25: Design a program to connect to access database and display contents of the tab

Theory:
Java with Access database integration typically involves using JDBC (Java Database Connectivity) to establish a connection between Java applications and Microsoft Access databases, enabling operations like querying, updating, and inserting data.

UCanAccess is a widely used JDBC driver for Access databases in Java, providing seamless integration and allowing developers to perform database operations using standard JDBC APIs.

By leveraging JDBC and UCanAccess, Java developers can create robust applications that interact with Access databases, offering functionalities ranging from data retrieval to complex database manipulation, fostering efficient data management solutions.

Learning Outcome:
We learnt how we can connect our MS access database with java for seamless access to data stored in the database through java programming.