

**Department of Computer Science**

**Lab Manual**

**SE231-L Object Oriented Programming**

Instructor’s Name: Kamran Khan

Student’s Name: Kabeer Abdul Karim

Roll No.: 67517 Batch: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Semester: FALL Year: 2024

Department: Computer Science

**Department of Computer Science**

**Lab Manual**

**SE231-L Object Oriented Programming**

Prepared By:

Ms. Umat Ul Shaiya

Reviewed / Approved By:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Faculty of Engineering Sciences & Technology**

**Iqra University**

CONTENTS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lab. No. | Date | a)List of Experiment | Total Marks | Signature | Page # |
| 1. |  | a) To install JDK and Eclipse  b) To learn Input/Output handling on Java console. Understanding variables using primitive and non-primitive data types. Exploring Java’s built-in classes. |  |  |  |
| 2. |  | To understand the control statements of Java including Loops & if-else. Exploring different operators used in Java. |  |  |  |
| 3. |  | To understand concepts of class and object in java. Implementing a class with members including data, methods and constructors. |  |  |  |
| 4. |  | To understand concepts method and constructor overloading. Learn how to provide different access controls on class members. |  |  |  |
| 5. |  | To understand Arrays, array index, single and multi-dimensional arrays, traversing the array using loop. Getting familiar with the String class of Java. |  |  |  |
| 6. |  | To understand the concept of inheritance, the superclass and subclass. |  |  |  |
| 7 |  | a) To understand the abstract method & class and final method and class.  b) Open Ended Lab |  |  |  |
|  |  | **Mid Term Examination** |  |  |  |
| 9 |  | a) To understand the concept packages & interfaces of Java.  b) To understand how runtime errors are being handled in Java. |  |  |  |
| 10. |  | To understand the design principles of graphical user interfaces (GUIs) using layout managers to arrange GUI components.  Understand basic component of JavaFx and their interaction used in different program of Java, such as Label, Button, Text Box, Combo Box etc. |  |  |  |
| 11. |  | To show different JavaFx Layouts and Charts. |  |  |  |
| 12. |  | Design a Login page using JavaFx components. |  |  |  |
| 13. |  | To study Data storage and retrieval in Signup page using Filing. |  |  |  |
| 14. |  | To understand the basics of class diagram of UML. |  |  |  |
| 15. |  | Assessment of Open-ended lab |  |  |  |
|  |  | Final Examination |  |  |  |

**Psychomotor Rubrics for Hardware based Lab**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations (>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Experimental Setup** | Able to setup experiment independently with complete understanding of each step | Able to setup experiment independently with adequate understanding of each step | Can setup major part of the experiment with assistance | Can’t set up the experiment even with assistance |
| **Procedure** | Able to follow the procedure completely with simplification or develop alternate  procedure | Able to follow the procedure completely | Able to follow major part of the procedure with errors or omissions | Unable to follow the procedure |
| **Experimental Results** | Able to achieve all the desired results with alternate ways to improve measurements | Able to achieve all the desired results | Able to achieve most of the desired results with errors | Unable to achieve the desired results |
| **Laboratory Manual** | All sections of the report are very well written and technically accurate. | All sections of the report are technically accurate. | Few sections of the report contain technical errors. | All sections of the report contain multiple technical errors. |

**Psychomotor Rubrics Assessment Hardware Lab**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Lab # | Score Allocation | | | | | | |
| **Experimental Setup**  **Marks (3)** | | **Procedure**  **Marks (2)** | **Experimental Results**  **Marks (3)** | | **Laboratory Manual**  **Marks (2)** | **Total Marks (10)** |
| 1 |  | |  |  | |  |  |
| 2 |  | |  |  | |  |  |
| 3 |  | |  |  | |  |  |
| 4 |  | |  |  | |  |  |
| 5 |  | |  |  | |  |  |
| 6 |  | |  |  | |  |  |
| 7 |  | |  |  | |  |  |
| 8 |  | |  |  | |  |  |
| 9 |  | |  |  | |  |  |
| 10 |  | |  |  | |  |  |
| 11 |  | |  |  | |  |  |
| 12 |  | |  |  | |  |  |
| 13 |  | |  |  | |  |  |
| 14 |  | |  |  | |  |  |
| Total Mark | | 140 | | | Total Obtained Marks | |  |

***Overall Score: \_\_\_\_\_\_\_\_\_\_\_out of 14 Examined by****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***(Obtained Score / 140) x 14 (Name and Signature of lab instructor)***

**Psychomotor Rubrics for Software based Lab**

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations (>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Software Skills** | Ability to use software with its standard and  advanced features without assistance | Ability to use software with its standard and  advanced features with minimal assistance | Ability to use software with its  standard features with assistance | Unable to use the software |
| **Programming/ Simulation** | Ability to program/ simulate the lab tasks  with simplification | Ability to program/ simulate the lab tasks  without errors | Ability to program/ simulate lab tasks  with errors | Unable to program/simulate |
| **Results** | Ability to achieve all the desired results with alternate ways | Ability to achieve all the desired results | Ability to achieve most of the desired results with errors | Unable to achieve the desired results |
| **Laboratory Manual** | All sections of the report are very well written and technically accurate. | All sections of the report are technically accurate. | Few sections of the report contain technical errors. | All sections of the report contain multiple technical errors. |

**Psychomotor Rubrics Assessment Software based Lab**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Lab # | Score Allocation | | | | | |
| **Software Skills**  **Marks (3)** | **Programming/ Simulation**  **Marks (2)** | **Experimental Results**  **Marks (3)** | | **Laboratory Manual**  **Marks (2)** | **Total Marks (3)** |
| 1 |  |  |  | |  |  |
| 2 |  |  |  | |  |  |
| 3 |  |  |  | |  |  |
| 4 |  |  |  | |  |  |
| 5 |  |  |  | |  |  |
| 6 |  |  |  | |  |  |
| 7 |  |  |  | |  |  |
| 8 |  |  |  | |  |  |
| 9 |  |  |  | |  |  |
| 10 |  |  |  | |  |  |
| 11 |  |  |  | |  |  |
| 12 |  |  |  | |  |  |
| 13 |  |  |  | |  |  |
| 14 |  |  |  | |  |  |
| Total Marks | | 140 | | Total Obtained marks | |  |

***Overall Score: \_\_\_\_\_\_\_\_\_\_\_out of 14 Examined by****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***(Obtained Score / 140) x 14 (Name and Signature of lab instructor)***

**Affective Domain Rubrics Assessment**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CATEGORY** | **Excellent (100% - 85%)** | **Good (84% - 75%)** | **Fair (74% - 60%)** | **Poor (Less than 60%)** |
| **Speaks Clearly** | Speaks clearly and distinctly all the time, and confidently. | Speaks clearly and distinctly most of the time, but is confused for a brief period of time, however, recovers. | Speaks clearly and distinctly most of the time, but seems not confident about what has been delivered. Shows lack of confidence. | Often mumbles or cannot be understood and clearly lacks confidence in delivering the content |
| **Points:** |  |  |  |  |
| **Preparedness** | Student is completely prepared and has obviously rehearsed. | Student seems pretty prepared but might have needed a couple more rehearsals. | The student is somewhat prepared, but it is clear that rehearsal was lacking. | Student does not seem at all prepared to present. |
| **Points** |  |  |  |  |
| **Answer back** | Student calmly listens to the questions and responds to the question confidently and correctly | Student calmly listens to the questions, responds confidently but some of the responses are incorrect. | Student shows anxiety while listening to the questions, and gives some correct responses, but some of the responses are incorrect. | Student shows anxiety while listening to the questions, and most of the responses are incorrect. |
| **Points:** |  |  |  |  |
| **Posture, Eye Contact & Speaking Volume** | Stands up straight, looks relaxed and confident. Establishes eye contact with everyone in the room during the presentation. Volume is loud enough to be heard by all members in the audience throughout the presentation. | Stands up straight and establishes eye contact with everyone in the room during the presentation. Volume is loud enough to be heard by the audience, but is sometimes not audible. | Sometimes stands up straight and establishes eye contact. Volume is loud enough to be heard by the audience, but many sentences spoken are not clear. | Lazy and informal posture. Does not look at people during the presentation. Volume is also too soft to be heard by the audience. |
| **Points:** |  |  |  |  |

***Overall Score: \_\_\_\_\_\_\_\_\_\_\_out of 14 Examined by****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***(Name and Signature of lab instructor)***

***Open Ended Lab Assessment Rubrics***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria and Scales** | | | |
| **Excellent**  **(10-8)** | **Good**  **(7-5)** | **Average**  **(4-2)** | **Poor**  **(1-0)** |
| **Criterion 1:** **Understanding the Problem:** How well the problem statement is understood by the student | | | |
| Understands the problem clearly and clearly identifies the underlying issues. | Adequately understands the problem and identifies the underlying issues. | Inadequately defines the problem and identifies the underlying issues. | Fails to define the problem adequately and does not identify the underlying issues. |
| **Criterion 2: Research:** The amount of research that is used in solving the problem | | | |
| Contains all the information needed for solving the problem | Good research, leading to a successful solution | Mediocre research which may or may not lead to an adequate solution | No apparent research |
| **Criterion 3: Class Diagram:** The completeness of the class diagram | | | |
| Class diagram with complete notations | Class diagram with incomplete notations | Class diagram with improper naming convention and notations | No Class diagram |
| **Criterion 4: Code:** How complete and accurate the code is along with the assumptions | | | |
| Complete Code according to the class diagram of the given case with clear assumptions | Incomplete Code according to the class diagram of the given case with clear assumptions | Incomplete Code according to the class diagram of the given case with unclear assumptions | Wrong code and naming conventions |
| **Criterion 5: Report:** How thorough and well organized is the solution | | | |
| All the necessary information clearly organized for easy use in solving the problem | Good information organized well that could lead to a good solution | Mediocre information which may or may not lead to a solution | No report provided |

***Open Ended Lab Assessment Rubrics***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria and Scales** | | | | |
| **Excellent**  **(10-8)** | **Good**  **(7-5)** | **Average**  **(4-2)** | **Poor**  **(1-0)** | **Total Marks**  **10** |
| **Criterion 1:**  **Understanding the Problem:** How well the problem statement is understood by the student | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| **Criterion 2:**  **Research**: The amount of research that is used in solving the problem | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| **Criterion 3:**  **Class Diagram:** The completeness of the class diagram | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| **Criterion 4:**  **Code:** How complete and accurate the code is along with the assumptions | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| **Criterion 5:**  **Report**: How thorough and well organized is the solution | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| Total | | | | **(\_\_\_\_/5)** |

Total marks obtained: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name and Signature of lab instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Rubrics for Lab Project / CCA**

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations**  **(>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Project Presentation + Project Demonstration** | Ability to demonstrate the project with achievement of required objectives having clear understanding of project limitations and future enhancements. Hardware and/or Software modules are fully functional, if applicable. | Ability to demonstrate the project with achievement of required objectives but understanding of project limitations and future enhancements is insufficient.  Hardware and/or Software modules are functional, if applicable. | Ability to demonstrate the project with achievement of a\*t least 50% required objectives and insufficient understanding of project limitations and future enhancements.  Hardware and/or Software modules are partially functional, if applicable. | Ability to demonstrate the project with achievement of less than 50% required objectives and lacks in understanding of project limitations and future enhancements.  Hardware and/or Software modules are not functional, if applicable. |
| **Project Report** | All sections of the Project report are very well- written and technically accurate. | All sections of the Project report are technically accurate. | Few sections of the Project report contain technical errors. | Project report has several grammatical/ spelling errors and sentence construction is poor. |
| **Viva** | Able to answer the questions easily and  correctly across the project. | Able to answer the questions related to the project | Able to answer the questions but with mistakes | Unable to answer the questions |

Total marks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name and Signature of lab instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Project / CCA Rubric based Assessment**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project # | Score Allocation | | | |
| **Project Presentation + Project Demonstration**  **Marks (5)** | **Project Report**  **Marks (3)** | **Viva**  **Marks (3)** | **Total**  **Marks (10)** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| Total Obtained Score | | | |  |

Total marks obtained: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name and Signature of lab instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Mid Term Rubrics**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations (>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Performance** | Able to present full knowledge of both problem and solution. | Able to present adequate knowledge of both problem and  solution | Able to present sufficient knowledge of both problem and solution | No or very less knowledge of both problems  and solution |
| **Viva** | Able to answer the questions easily and  correctly | Able to answer the questions | Able to answer the questions but with mistakes | Unable to answer the questions |

***Mid Term Rubrics based Assessment***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester, Batch: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |
| --- | --- |
| **Score Allocation** | |
| **Performance** | \_\_\_\_\_\_ /20 |
| **Viva** | \_\_\_\_\_ /5 |
| **Total Obtained Score** | **\_\_\_\_\_\_/ 25** |

***Examined by****:*

***(Name and Signature of concerned lab instructor***

***Final Term Rubrics***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations (>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Performance** | Able to present full knowledge of both problem and solution. | Able to present adequate knowledge of both problem and  solution | Able to present sufficient knowledge of both problem and solution | No or very less knowledge of both problems  and solution |
| **Viva** | Able to answer the questions easily and  correctly | Able to answer the questions | Able to answer the questions but with mistakes | Unable to answer the questions |

***Final Term Rubrics based Assessment***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester, Batch: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |
| --- | --- |
| Score Allocation | |
| **Performance** | \_\_\_\_\_ /45 |
| **Viva** | \_\_\_\_\_ /5 |
| **Total Obtained Score** | **\_\_\_\_\_/ 50** |

***Examined by****:*

***(Name and Signature of concerned lab instructor)***

**Final Lab Assessment**

|  |  |  |  |
| --- | --- | --- | --- |
| **Assessment Tool** | **CLO-1 (20)** | **CLO-2 (20)** | **CLO-3 (10)** |
| **Lab Manual** |  |  |  |
| **Subject Project / Viva** |  |  |  |
| **Lab Exam / Viva** |  |  |  |
| **Score Obtained** |  |  |  |
| **Total Score: \_\_\_\_\_\_\_\_ out of 50** | | | |

***Examined by****:*

***(Name and Signature of concerned lab instructor)***

**Lab Session 1**

|  |
| --- |
| **Objective:**   1. Getting familiar with the Java development kit (JDK). Running your first Java program using CMD and an IDE. 2. To learn Input/output handling on Java console. Understanding variables using primitive and non-primitive data types. Exploring Java’s built-in classes.   **Introduction:** |

**What is JDK?**It's the full featured Software Development Kit for Java, including JRE, and the compilers and tools (like Java Debugger) to create and compile programs.  
JRE is required to run Java programs while JDK is required when you have to do some Java programming.

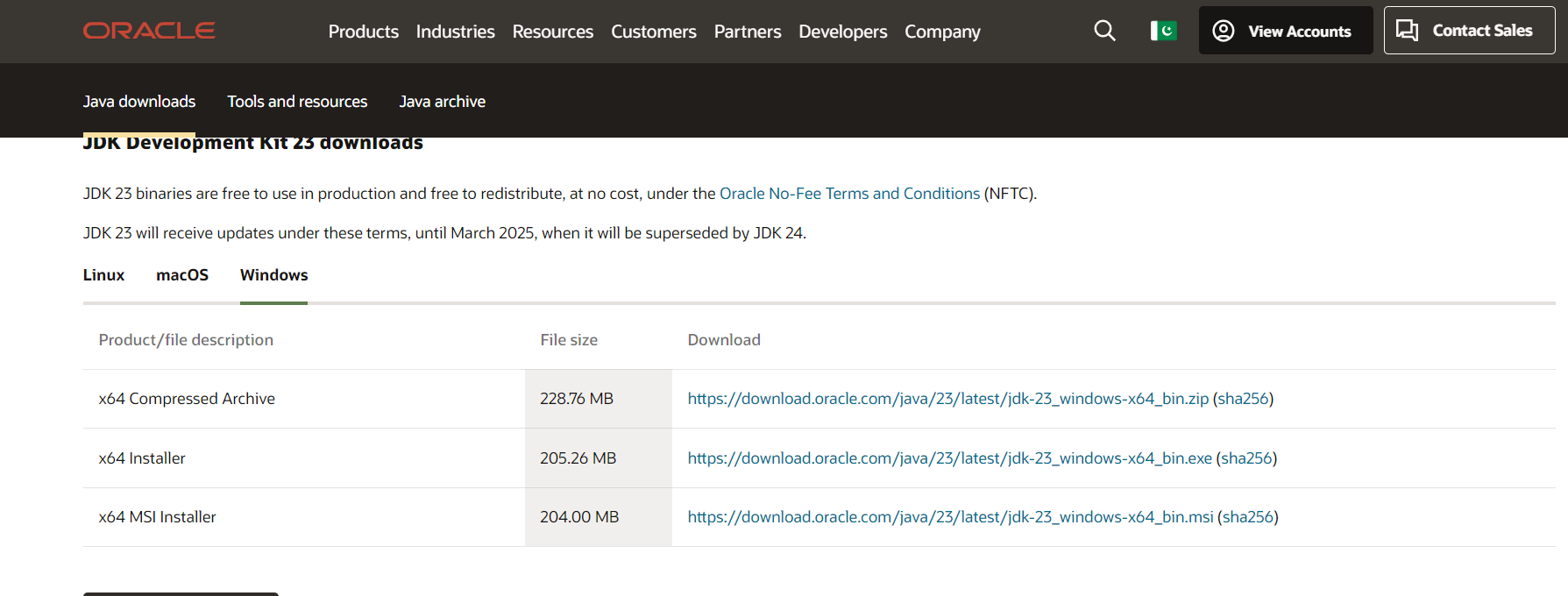
**Installing JDK for Windows**

1. Download the JDK from Oracle’s website. Choose the right JDK depending upon your system’s specifications.

**Required IDE:**

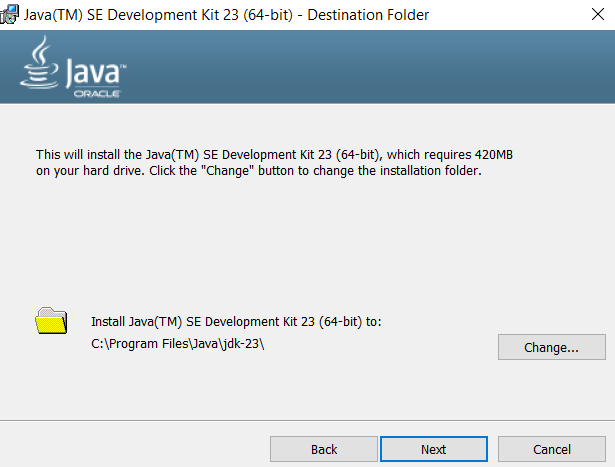
* JDK (Java Development Kit)
* Eclipse

#### **Procedure:** **Step 1: Setup the JDK:**

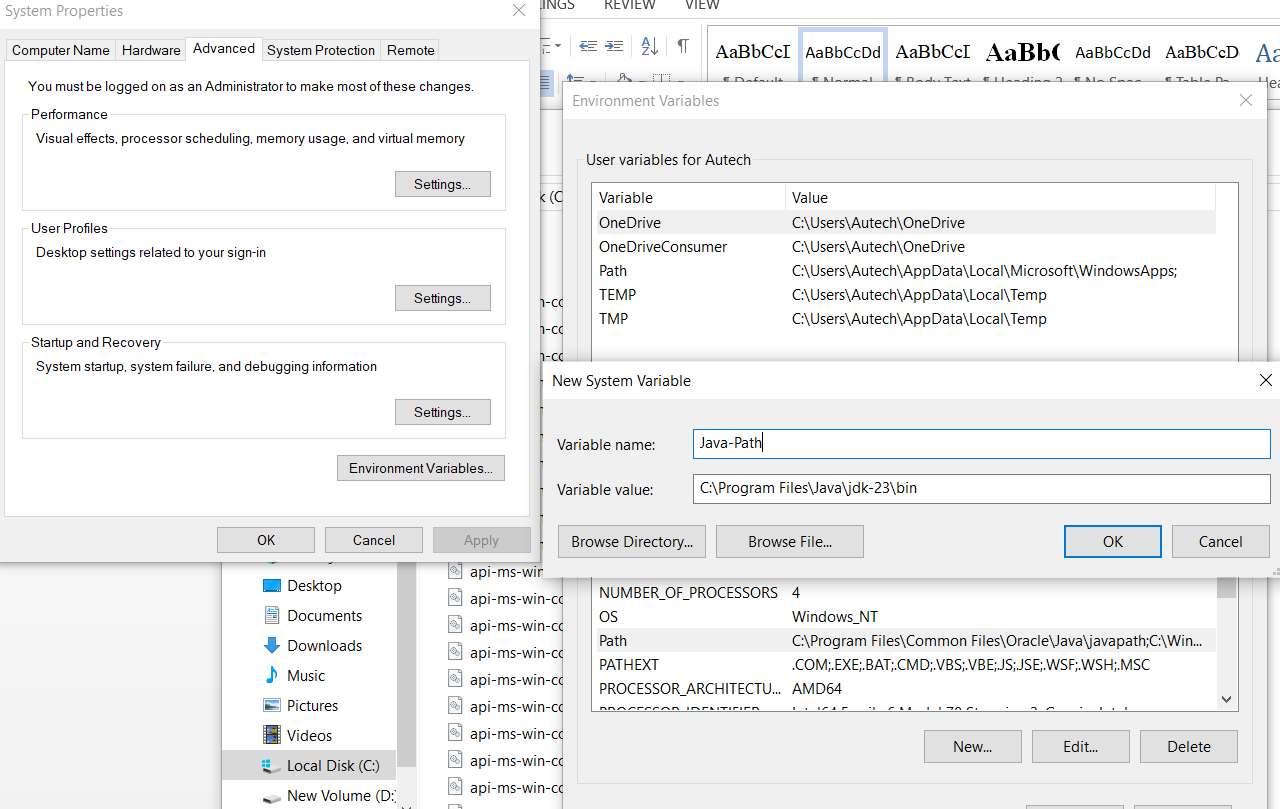


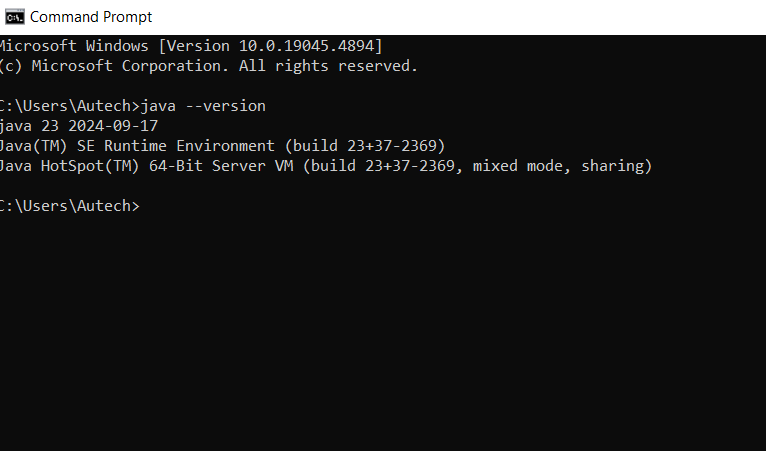






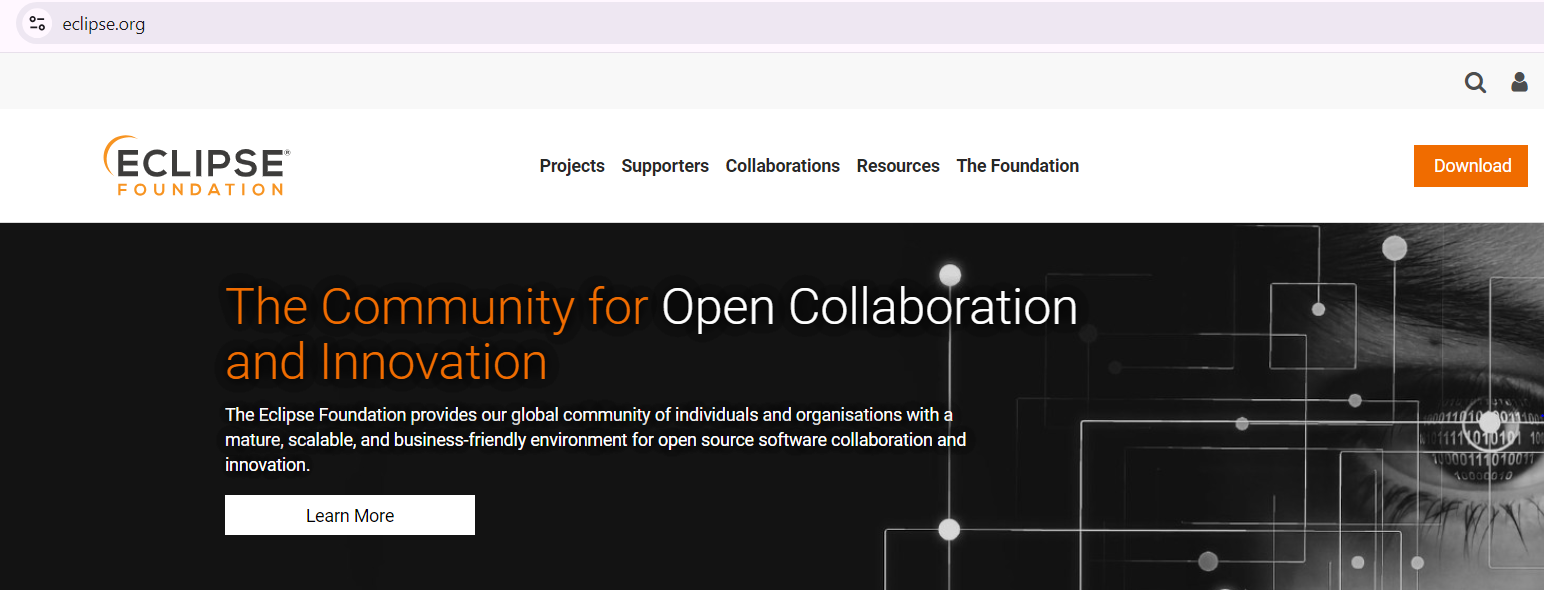


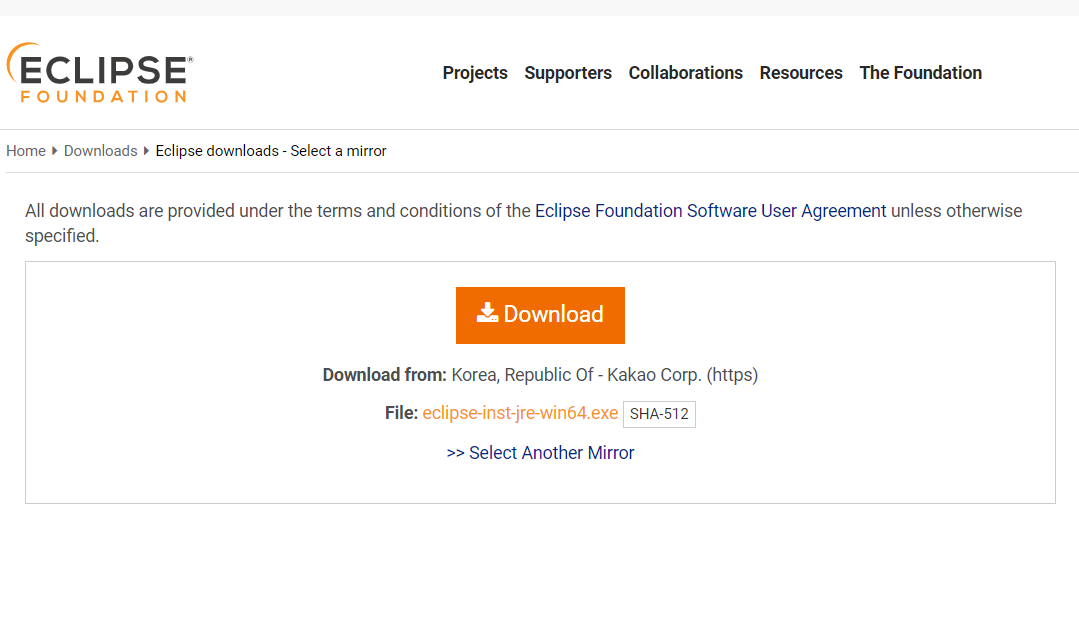




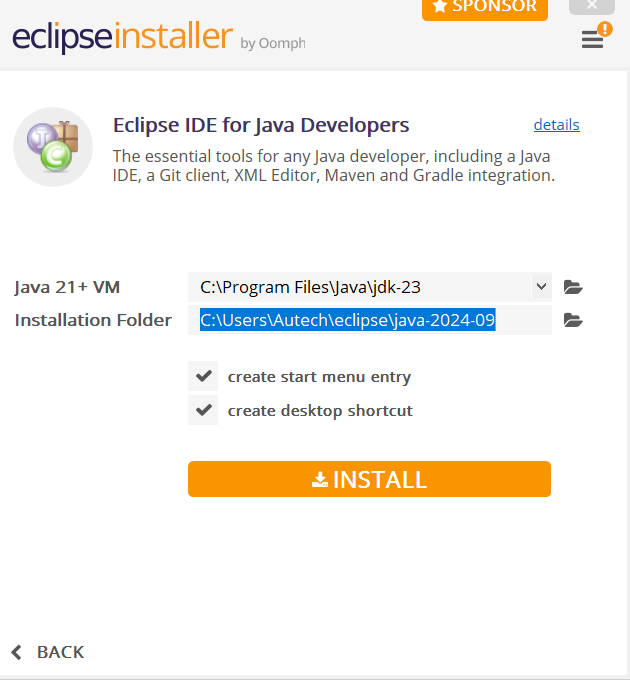
#### **Step 2: Setup the Eclipse IDE**

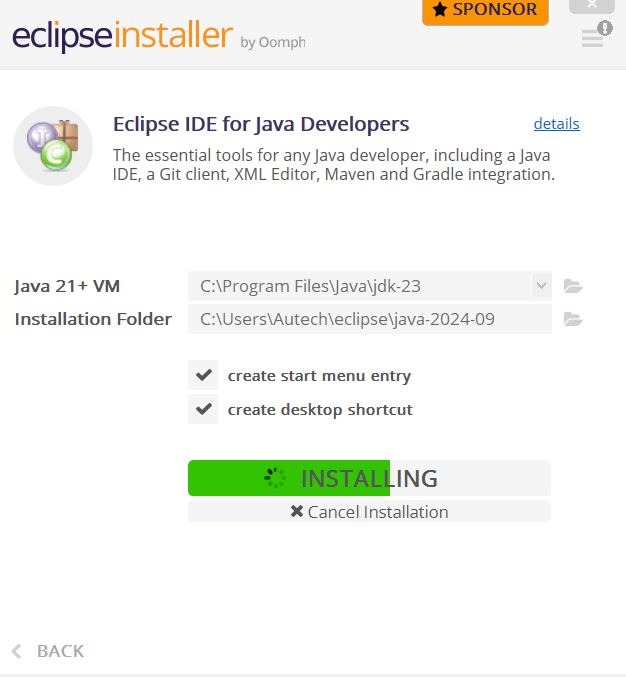
1. **Install IDE**: Ensure that your IDE is installed and configured properly with JDK (Java Development Kit).
2. Download Eclipse form its official website and install it on your system by following the instruction provided by the installer.



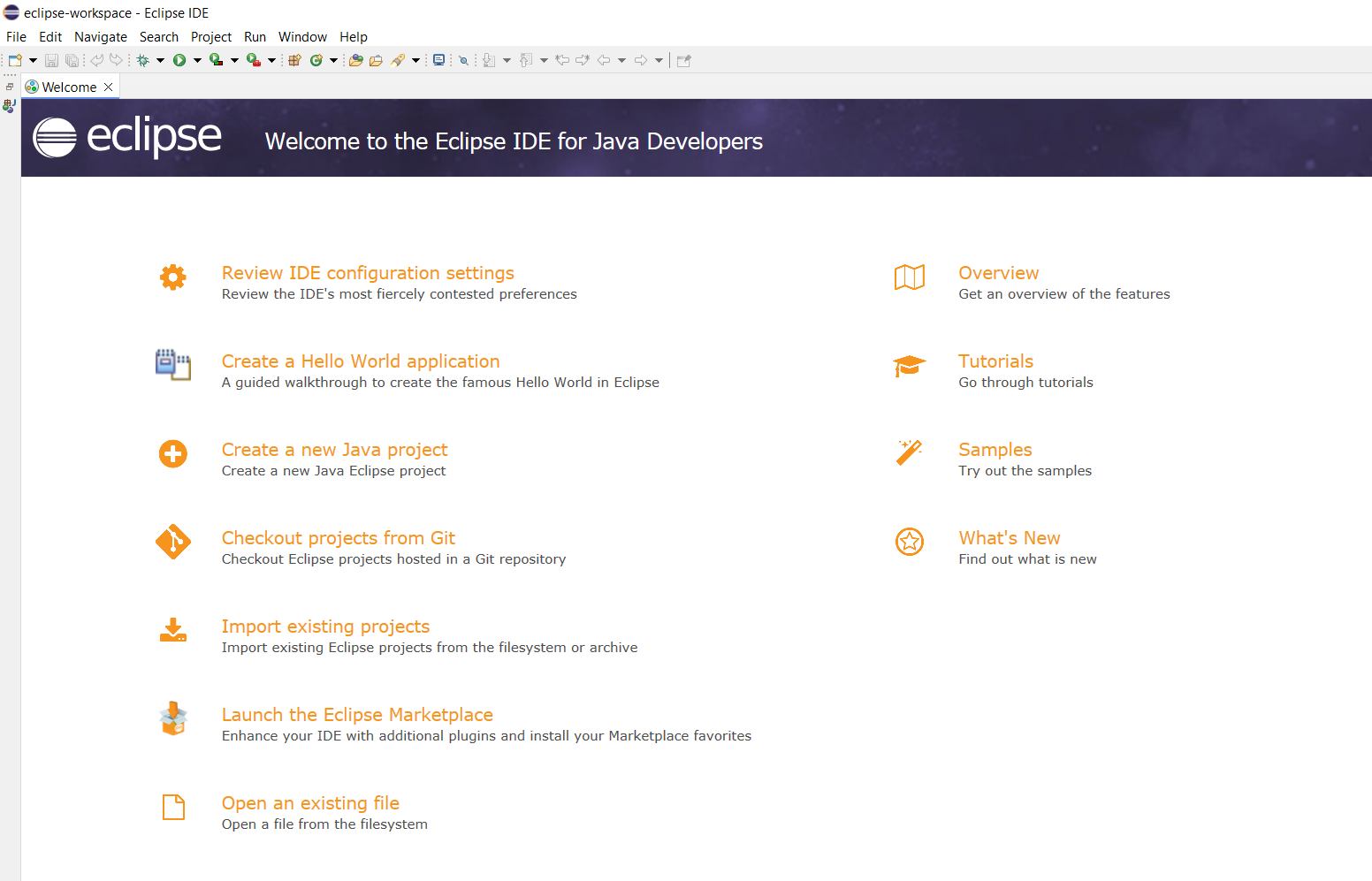


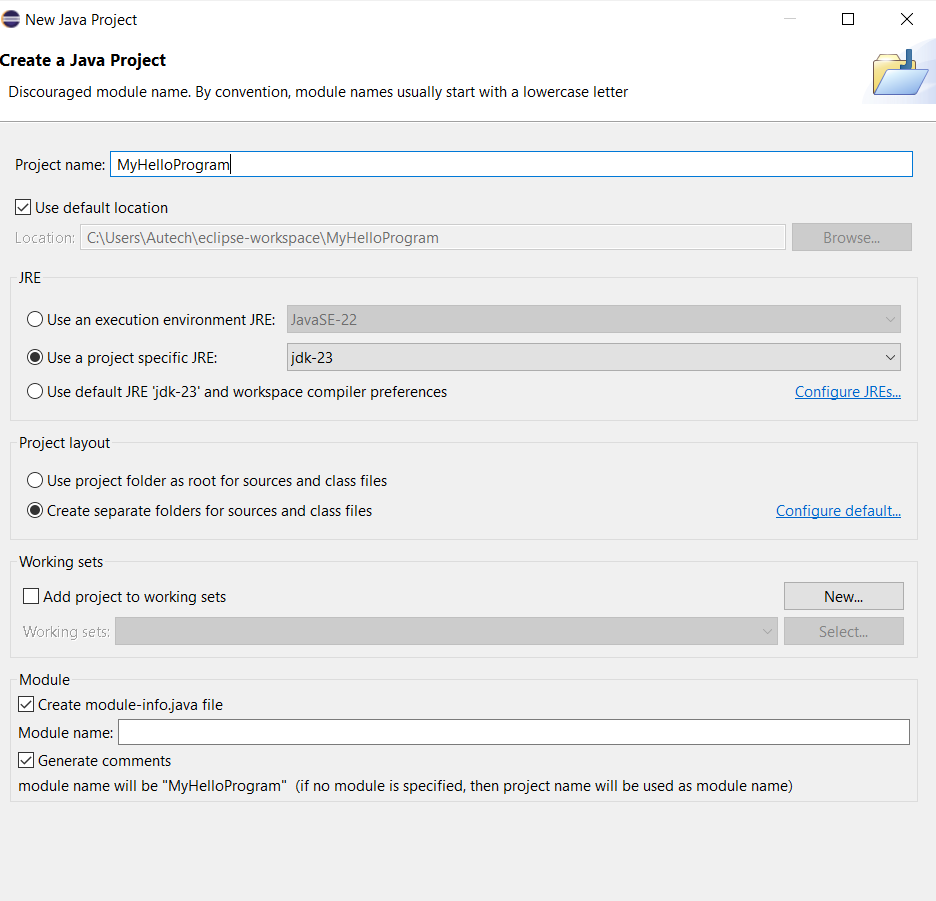


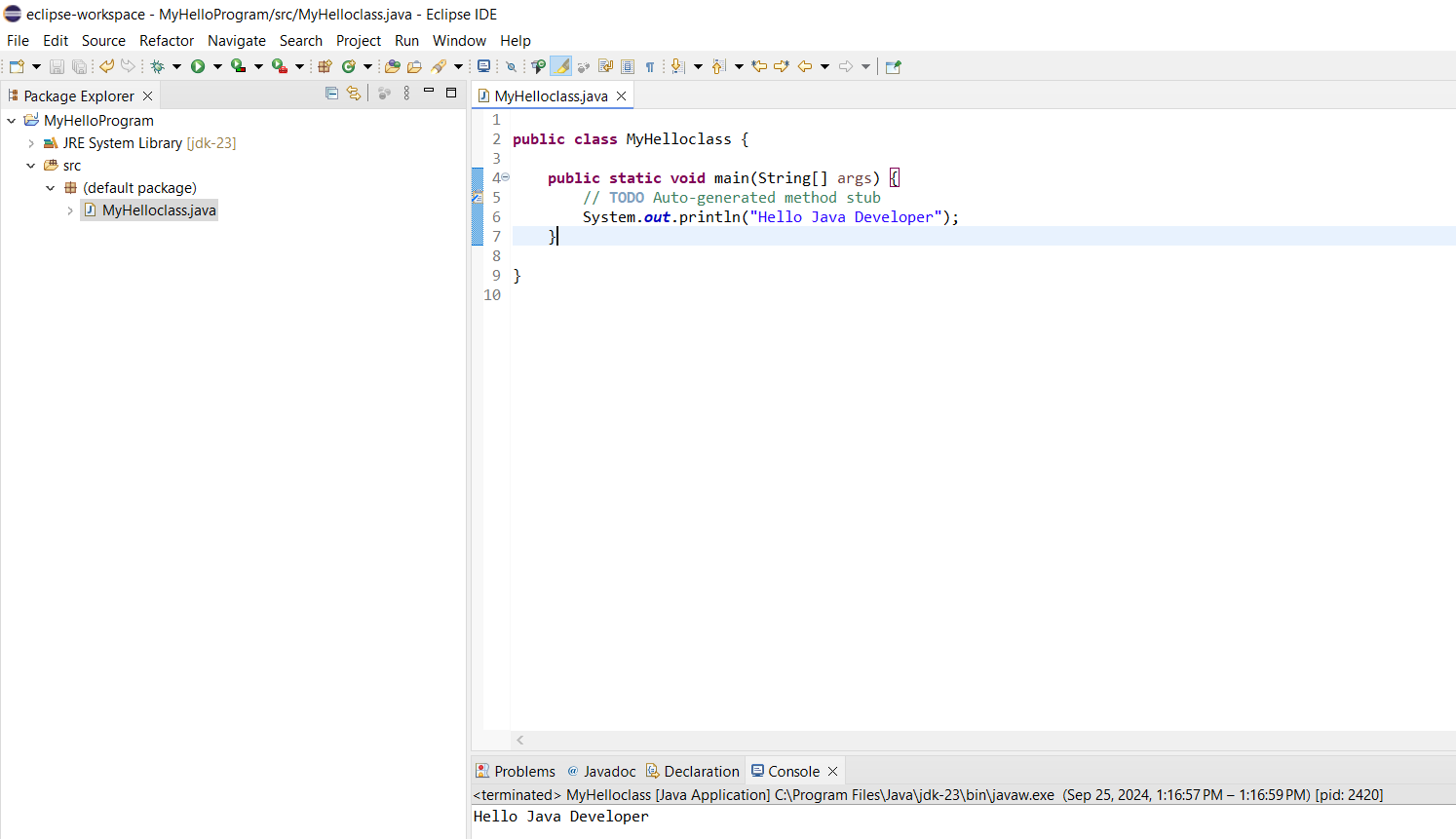












#### **Understanding Java Console Input and Output**

**Theory:**

**Console input**

System.in to the standard input device. Console input is not directly supported in Java, but Scanner class is used to create an object to read input from System.in, as follows:

Scanner input = new Scanner(System.in);

double radius = input.nextDouble();

Import the class by adding

import java.util.Scanner;

**Console output**

Java uses System.out to refer to the standard output device.To perform console output, println method is used to display a primitive value or a string to the console.

System.out.print("Hello ");

System.out.println("world");

You can use the System.out.printf method to display formatted output on the console.

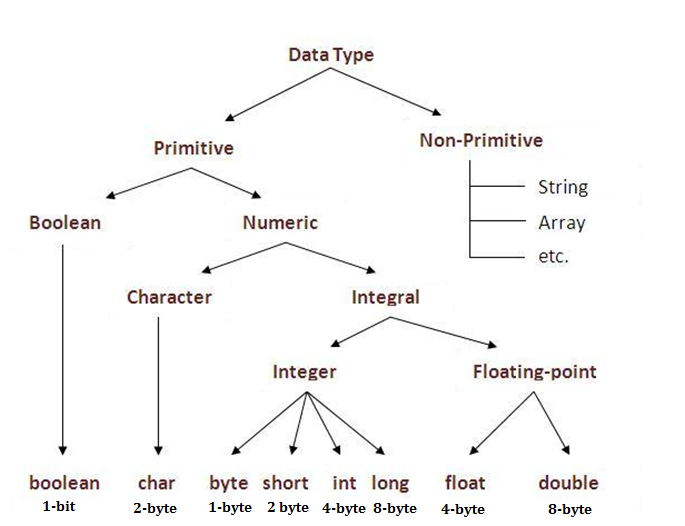
System.out.printf(“Your Total amount is %4.2f", total);

System.out.printf("count is %d and amount is %f", count, amount);

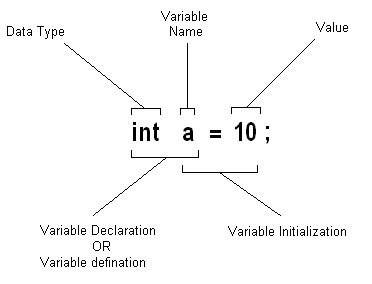
**Data Types in Java**

A data type in a programming language is a set of data with values having predefined characteristics.

There are two basic types in Java.

1. **Primitive**  
    A primitive type is predefined by the language and is named by a reserved keyword.
2. **Non-Primitive**It is a reference data type, which are references to objects.

**Figure 2.1: Data Types in Java**

**Variables**

Variable is a name of memory location.

It is name of reserved area allocated in memory.

In the given example; int is data type, a is variable name

and 10 is the value that a variable holds, followed by a terminator;

**Figure 2.2: Variable Initialization**

**Type Conversion**

Casting is an operation that converts a value of one data type into a value of another data type. The syntax for casting a type is to specify the target type in parentheses, followed by thevariable’s name or the value to be cast. For example;

System.out.println((int)1.7);

The above statement displays 1. When a double value is cast into an int value, the fractional part is truncated.

**Some Useful Java Classes**

**Math**

Math class file is included for the definitions of math functions listed below. It is written as java.lang.Math.

Trignometic / Maths Functions

|  |  |
| --- | --- |
| * sin(n) * cos(n) * tan(n) * sinh(n) | * hosh(n) * tanh(n) * pow(nmb,pwr) * sqrt(n) |

**Date**

Java provides a system-independent encapsulation of date and time in the java.util.Date class. The no-arg constructor of the Date class can be used to create an instance for the current date and time.

1. **Write a Java program to explore Math class.**

public class MathClass

{

public static void main(String[] args)

{

double a=45,b=1,sn,cs,tn,snh,csh,tnh;

sn=Math.sin(a);

cs=Math.cos(a);

tn=Math.tan(a);

snh=Math.sinh(b);

csh=Math.cosh(b);

tnh=Math.tanh(b);

System.out.println("\nTrignometric Functions");

System.out.println("sin 45 = " + sn);

System.out.println("cos 45 =" + cs);

System.out.println("tan 45 =" + tn);

System.out.println("\nHyperbolic Functions");

System.out.println("sinh 1 = " + snh);

System.out.println("cosh 1 = " + csh);

System.out.println("tanh 1 = " + tnh);

|  |
| --- |
| **Expected Outcome:** |

**Task #1**

**Scenario:** You are building a simple Java console application that asks the user for their first name, last name, and age. Once the user provides the input, the program should display a welcome message that includes their full name and their age in 5 years.

**Task Description:**   
Write a Java program that handles user input from the console for first name, last name, and age. Then, output a welcome message that includes their full name and calculates their age in 5 years.

|  |
| --- |
| **Code:**  import java.util.Scanner;  class Main { // Changed from "public class WelcomeApp"  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter your first name: ");  String firstName = scanner.nextLine();  System.out.print("Enter your last name: ");  String lastName = scanner.nextLine();  System.out.print("Enter your age: ");  int age = scanner.nextInt();  int ageIn5Years = age + 5;  System.out.println("\nWelcome, " + firstName + " " + lastName + "!");  System.out.println("In 5 years, you will be " + ageIn5Years + " years old.");  scanner.close();  }}  **OUTPUT:** |

**Task #2**

**Scenario:** Create a basic program for a fitness tracking app that computes the total distance a person runs in a week. The app keeps track of the distance the user runs each day, from Monday to Sunday.

1. Define seven variables (of type double) to hold the distance the user runs for each day of the week: monday, tuesday, wednesday, thursday, friday, saturday, and sunday.
2. Declare another variable, totalDistance, to store the total distance run over the week.
3. Ask the user to input the distance they ran each day and save the input in the corresponding variables.
4. The total distance run for the week will be calculated by adding up all the daily distances and saving it in totalDistance.
5. Show the total distance to the user.

**Task Description:** Develop a fitness tracking app that computes the total distance a person runs over a week by storing daily running distances in seven variables (double type) for each day from Monday to Sunday. Prompt the user for the daily inputs and calculate the total distance by summing up all the daily distances. Display the total distance to the user at the end of the program.

|  |
| --- |
| **Code:**  import java.util.Scanner;  public class Student {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          double monday, tuesday, wednesday, thursday, friday, saturday, sunday;          double totalDistance;          System.out.print("Enter distance run on Monday (km): ");          monday = scanner.nextDouble();          scanner.nextLine();          System.out.print("Enter distance run on Tuesday (km): ");          tuesday = scanner.nextDouble();          scanner.nextLine();          System.out.print("Enter distance run on Wednesday (km): ");          wednesday = scanner.nextDouble();          scanner.nextLine();          System.out.print("Enter distance run on Thursday (km): ");          thursday = scanner.nextDouble();          scanner.nextLine();          System.out.print("Enter distance run on Friday (km): ");          friday = scanner.nextDouble();          scanner.nextLine();          System.out.print("Enter distance run on Saturday (km): ");          saturday = scanner.nextDouble();          scanner.nextLine();          System.out.print("Enter distance run on Sunday (km): ");          sunday = scanner.nextDouble();          scanner.nextLine();            totalDistance = monday + tuesday + wednesday + thursday + friday + saturday + sunday;            System.out.println("\nTotal distance run this week: " + totalDistance + " km");          scanner.close();      }  }  **OUTPUT:** |

**Task #3**

**Scenario:** You are working on an e-commerce platform that needs a feature to calculate discounts on products and display the current date of the transaction.

**Task Description:** Create a program that takes the product's original price, applies a discount percentage, and calculates the final price after the discount. The program should also display the current date and time of the transaction.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 2**

**Objective:**

To understand Java control statements, including loops and if-else structures, and to explore various operators.  
**Required Tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

In this lab, we will focus on Java control statements which allow for decision-making and repetition in code. Control statements like if-else, switch, and loops (for, while, do-while) are integral in controlling the program's flow. Along with control statements, operators like arithmetic, relational, logical, and bitwise will also be explored. Through practical tasks, we will reinforce the theoretical knowledge of control structures and operators.

### Java Control Statements:

1. **If-Else Structure**
   * Allows conditional execution of code blocks.
   * Syntax:

if (condition) {

// code executed if condition is true

} else {

// code executed if condition is false

}

1. **Switch Case**
   * An alternative to the if-else-if ladder, used when there are multiple possible values for a single variable.
   * Syntax:

switch (expression) {

case value1:

// code for value1

break;

case value2:

// code for value2

break;

default:

// code for no matches

}

1. **Loops**
   * **For Loop**: Executes a block of code a specific number of times.

for (initialization; condition; increment/decrement) {

// code to be repeated

}

* + **While Loop**: Repeats a block of code while a condition is true.

while (condition) {

// code to be repeated

}

* + **Do-While Loop**: Executes the block at least once before checking the condition.

do {

// code to be repeated

} while (condition);

**Procedure:**

#### **Using if-else-if Ladder to Determine the Season**

Write a Java program that takes the current month as an integer (1 for January, 12 for December) and prints the corresponding season. Use an if-else-if ladder.

import java.util.Scanner;

public class SeasonFinder {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter month (1-12): ");

int month = input.nextInt();

if (month == 12 || month == 1 || month == 2) {

System.out.println("Season: Winter");

} else if (month >= 3 && month <= 5) {

System.out.println("Season: Spring");

} else if (month >= 6 && month <= 8) {

System.out.println("Season: Summer");

} else if (month >= 9 && month <= 11) {

System.out.println("Season: Fall");

} else {

System.out.println("Invalid month entered.");

}

}

}

#### **Generate a Number Pattern using Nested Loops**

Create a Java program to generate the following pattern using nested loops:

|  |
| --- |
| 1  12  123  1234  12345 |

java

Copy code

public class NumberPattern {

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

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

System.out.print(j);

}

System.out.println();

}

}

}

#### **Fibonacci Sequence Generator**

Write a program to generate the first n numbers in the Fibonacci sequence.

import java.util.Scanner;

public class FibonacciGenerator {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter the number of Fibonacci numbers to generate: ");

int n = input.nextInt();

int first = 0, second = 1;

System.out.print("Fibonacci Series: " + first + ", " + second);

for (int i = 3; i <= n; i++) {

int next = first + second;

System.out.print(", " + next);

first = second;

second = next;

}

}

}

#### **Prime Number Finder**

Write a Java program to check if a number is prime or not.

import java.util.Scanner;

public class PrimeChecker {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter a number to check if it's prime: ");

int num = input.nextInt();

boolean isPrime = true;

if (num <= 1) {

isPrime = false;

} else {

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

isPrime = false;

break;

}

}

}

if (isPrime) {

System.out.println(num + " is a prime number.");

} else {

System.out.println(num + " is not a prime number.");

}

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

**Scenario:** You are developing a Java program that takes a user's input for their exam score and determines if they passed or failed. The passing score is 50 or above. Based on the score, the program should output either "Pass" or "Fail."

**Task Description:**  
Write a Java program that takes an exam score as input and uses an if-else structure with comparison operators to determine if the user passed or failed. Display an appropriate message based on the result.

|  |
| --- |
|  |

**Task #2**

**Scenario:** You are tasked with creating a Java program that calculates the sum of all even numbers between 1 and 100. The program should use a loop structure to iterate through the numbers and add the even ones to a running total.

**Task Description:**  
Write a Java program using a loop and arithmetic operators to calculate and print the sum of all even numbers from 1 to 100.

|  |
| --- |
|  |

**Task #3**

**Scenario:** You are developing a weather prediction system that advises users on how to dress based on the temperature and weather conditions. The system will provide recommendations based on the temperature (in degrees Celsius) and whether it is raining or sunny.

**Instructions:**

1. Prompt the user to enter the current temperature (as an integer) and whether it is raining or sunny (as a string, either "rainy" or "sunny").
2. Based on the temperature and weather conditions, the system should give advice according to the following logic:
   * If the temperature is **below 0°C**:
     + If it's **raining**, suggest: "Wear a heavy coat and take an umbrella."
     + If it's **sunny**, suggest: "Wear a heavy coat and sunglasses."
   * If the temperature is **between 0°C and 10°C**:
     + If it's **raining**, suggest: "Wear a warm jacket and take an umbrella."
     + If it's **sunny**, suggest: "Wear a warm jacket and sunglasses."
   * If the temperature is **between 11°C and 20°C**:
     + If it's **raining**, suggest: "Wear a light jacket and take an umbrella."
     + If it's **sunny**, suggest: "Wear a light jacket and sunglasses."
   * If the temperature is **above 20°C**:
     + If it's **raining**, suggest: "Wear light clothing and take an umbrella."
     + If it's **sunny**, suggest: "Wear light clothing and sunglasses."
3. Use if-else if statements to implement this logic.
4. After processing the input, display the appropriate advice to the user.

**Task Description:**  
You are building a weather prediction system that suggests what to wear based on the temperature and weather (rainy or sunny). The program asks the user to input the current temperature and weather condition, and then it gives advice on appropriate clothing. The recommendations change depending on whether it's cold or warm and if it's rainy or sunny.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 3**

**Objective:**

Understanding concepts of class and object in Java. Implementing a class with members including data, methods, and constructors.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

### **Class:** A class is a blueprint or template for creating objects. It defines the attributes (data) and behaviors (methods or functions) that the objects created from the class will have.

A class consists of:

* **Data (Variables):** Attributes that define the properties of the class.
* **Methods:** Functions defined within the class that operate on the class's data.
* **Constructors:** Special methods that are called when an object of the class is instantiated.

### **Objects in Java:**

An object is an instance of a class. It represents a specific entity that has the properties and behaviors defined by its class.

**Procedure:**

### **Box Class Implementation**

The following code demonstrates the creation of a Box class with methods and a demo class to calculate the volume of boxes.

// Box class definition

class Box {

double width;

double height;

double depth;

// Compute and return volume

double volume() {

return width \* height \* depth;

}

}

// \_\_\_\_\_\_\_\_\_\_\_ Demo Class \_\_\_\_\_\_\_\_\_\_\_

class BoxDemo4 {

public static void main(String args[]) {

Box mybox1 = new Box(); // Create first object

Box mybox2 = new Box(); // Create second object

double vol;

// Assign values to mybox1's instance variables

mybox1.width = 10;

mybox1.height = 20;

mybox1.depth = 15;

// Assign different values to mybox2's instance variables

mybox2.width = 3;

mybox2.height = 6;

mybox2.depth = 9;

// Get volume of first box

vol = mybox1.volume();

System.out.println("Volume of Box 1 is " + vol);

// Get volume of second box

vol = mybox2.volume();

System.out.println("Volume of Box 2 is " + vol);

}

}

### **Adding Constructor**

Now, we will add a constructor to the Box class that initializes the box dimensions to default values.

class Box {

double width;

double height;

double depth;

// This is the constructor for Box.

Box() {

System.out.println("Constructing Box");

width = 10;

height = 10;

depth = 10;

}

// Compute and return volume

double volume() {

return width \* height \* depth;

}

}

### **Create a Calculator Class**

Create a class Calculator with the following details:

* The class should contain default and parameterized constructors. The constructors should print the statements:
  + “Inside Default Constructor”
  + “Inside Parameterized Constructor”
* The class should contain the following data fields and methods:
  + int square = 2;
  + int cube = 3;
  + calculateSquare(int x)
  + calculateCube(int x)
  + calculateFactorial(int x)
  + generateTable(int x)
* Create objects of this class using both constructors in the main class.
* Call all four functions via the objects.

// Calculator class definition

class Calculator {

int square = 2;

int cube = 3;

// Default constructor

Calculator() {

System.out.println("Inside Default Constructor");

}

// Parameterized constructor

Calculator(int x) {

System.out.println("Inside Parameterized Constructor");

}

// Calculate square

int calculateSquare(int x) {

return x \* x;

}

// Calculate cube

int calculateCube(int x) {

return x \* x \* x;

}

// Calculate factorial

int calculateFactorial(int x) {

if (x == 0 || x == 1) {

return 1;

} else {

return x \* calculateFactorial(x - 1);

}

}

// Generate multiplication table

void generateTable(int x) {

System.out.println("Multiplication Table of " + x + ":");

for (int i = 1; i <= 10; i++) {

System.out.println(x + " x " + i + " = " + (x \* i));

}

}

}

// Main class to demonstrate Calculator

public class CalculatorDemo {

public static void main(String[] args) {

// Create object using default constructor

Calculator calcDefault = new Calculator();

// Create object using parameterized constructor

Calculator calcParam = new Calculator(5);

// Call methods

System.out.println("Square of 4: " + calcDefault.calculateSquare(4));

System.out.println("Cube of 3: " + calcDefault.calculateCube(3));

System.out.println("Factorial of 5: " + calcDefault.calculateFactorial(5));

calcDefault.generateTable(7);

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

**Scenario:** You are developing an **online shopping cart** system for an e-commerce website. Each item added to the cart is represented as an object of the Item class. You will use **constructors** to initialize the objects with the item details, such as item name, price, and quantity.

The system needs to calculate the total price of the items in the cart, display each item’s details, and update the quantity of an item if more units are added.

### Requirements:

1. **Item Class**: This class will represent the items that the user can add to the cart. Each item has:
   * name: The name of the item (String)
   * price: The price of the item (double)
   * quantity: The quantity of the item added to the cart (int)
2. **Constructor**: The Item class will use a **constructor** to initialize the attributes name, price, and quantity when a new item is added to the cart.
3. **Methods**:
   * A method getTotalPrice() that calculates and returns the total price of the item based on its quantity and price.
   * A method displayItemDetails() to display the item's details (name, price, quantity, and total price).
   * A method updateQuantity() to update the quantity of the item if more units are added.

**Task Description:** You are building an online shopping cart system where each product in the cart is represented as an item. The system uses a class called Item to store details like the item’s name, price, and quantity. A constructor initializes these values when a new item is added to the cart. The system can calculate the total price for each item, display the item’s details, and update the quantity if more units of the item are added.

|  |
| --- |
|  |

**Task #2**

**Scenario:** You need to create a Student class for a school application. The class should have the following attributes:

* String name
* int rollNumber
* double[] grades

Implement a constructor to initialize name and rollNumber, and create a method to calculate the average grade. Additionally, create a method to display the student’s details along with their average grade.

**Task Description:** What would be the implementation of the Student class, and how would you create instances of this class to track multiple students' grades?

|  |
| --- |
|  |

**Task #3**

**Scenario:** You are tasked with designing a Car class for a car showroom application. The class should include:

* String make
* String model
* int year
* double price

Implement a constructor that takes parameters for all these attributes. Add methods to:

* Display car details.
* Apply a discount on the car price.
* Check if the car is a classic (older than 20 years).

**Task Description:** How would you define the Car class and utilize it in a main method to showcase different car objects and their functionalities?

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 4**

**Objective:**

Understanding the concepts of **method overloading**, **constructor overloading**, and **access control** in Java. Learn how to define a class with multiple methods and constructors with different parameters and apply access controls to class members.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

### **Method Overloading**

Method Overloading occurs when multiple methods in a class share the same name but differ in the number or type of parameters. It allows methods to behave differently depending on the argument list.

### **Constructor Overloading**

Constructor Overloading involves defining multiple constructors with different parameters in a class, allowing objects to be instantiated in various ways.

### **Access Control**

In Java, access control defines the visibility and accessibility of class members (fields, methods, constructors). There are four types:

1. **private**: Accessible only within the class.
2. **default (package-private)**: Accessible within the same package.
3. **protected**: Accessible within the same package and by subclasses.
4. **public**: Accessible from anywhere.

**Procedure:**

### **Method Overloading Demonstration**

We will create a class OverloadDemo that demonstrates method overloading by having several methods named test but with different parameters.

class OverloadDemo {

// Method with no parameters

void test() {

System.out.println("No parameters");

}

// Overloaded method with one integer parameter

void test(int a) {

System.out.println("a: " + a);

}

// Overloaded method with two integer parameters

void test(int a, int b) {

System.out.println("a and b: " + a + " " + b);

}

// Overloaded method with one double parameter

double test(double a) {

System.out.println("double a: " + a);

return a \* a;

}

// No-arg constructor

OverloadDemo() {

System.out.println("No-args constructor");

}

// Parameterized constructor

OverloadDemo(int demo) {

System.out.println("Parameterized Constructor: " + demo);

}

}

class Overload {

public static void main(String[] args) {

OverloadDemo ob = new OverloadDemo(); // Default constructor

OverloadDemo ob1 = new OverloadDemo(33); // Parameterized constructor

double result;

// Call all versions of test()

ob.test();

ob.test(10);

ob.test(10, 20);

result = ob.test(123.25);

System.out.println("Result of ob.test(123.25): " + result);

}

}

### **Account Class Implementation**

We will implement the Account class with private data members and provide overloaded constructors, accessor, and mutator methods. We will also implement methods to calculate the monthly interest and modify the balance.

import java.util.Date;

class Account {

private int id;

private double balance;

private static double annualInterestRate;

private Date dateCreated;

// No-arg constructor

public Account() {

this.id = 0;

this.balance = 0;

this.dateCreated = new Date();

}

// Parameterized constructor

public Account(int id, double balance) {

this.id = id;

this.balance = balance;

this.dateCreated = new Date();

}

// Getter and setter for id

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

// Getter and setter for balance

public double getBalance() {

return balance;

}

public void setBalance(double balance) {

this.balance = balance;

}

// Getter and setter for annualInterestRate

public static double getAnnualInterestRate() {

return annualInterestRate;

}

public static void setAnnualInterestRate(double annualInterestRate) {

Account.annualInterestRate = annualInterestRate;

}

// Getter for dateCreated

public Date getDateCreated() {

return dateCreated;

}

// Method to calculate monthly interest rate

public double getMonthlyInterestRate() {

return annualInterestRate / 12;

}

// Method to calculate monthly interest

public double getMonthlyInterest() {

return balance \* getMonthlyInterestRate() / 100;

}

// Method to withdraw amount from the balance

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

}

}

// Method to deposit amount to the balance

public void deposit(double amount) {

balance += amount;

}

}

class AccountDemo {

public static void main(String[] args) {

// Create an Account object

Account account = new Account(1122, 20000);

Account.setAnnualInterestRate(4.5);

// Withdraw and deposit

account.withdraw(2500);

account.deposit(3000);

// Print account details

System.out.println("Balance: $" + account.getBalance());

System.out.println("Monthly Interest: $" + account.getMonthlyInterest());

System.out.println("Date Created: " + account.getDateCreated());

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

**Scenario:** You are developing a simple payment system for an e-commerce application. Create a class PaymentProcessor that processes payments in different ways:

* If no parameters are provided, it should print "Processing default payment."
* If one integer parameter is provided, it should print "Processing payment of amount [parameter]."
* If two parameters (an integer and a string) are provided, it should print "Processing payment of [parameter] for [customer name]."

**Task Description:** How would you implement the PaymentProcessor class with method overloading, and how would you call each overloaded method in the main class?

|  |
| --- |
|  |

**Task #2**

You are designing a **Library Management System** that keeps track of books in a library. The system uses a Book class that represents the books available. To manage different types of books, you will implement **constructor overloading** within the Book class to allow for various ways of creating book objects based on different parameters.

### Requirements:

1. **Book Class**: This class will represent the books in the library with the following attributes:
   * title: The title of the book (String)
   * author: The author of the book (String)
   * ISBN: The International Standard Book Number (String)
   * yearPublished: The year the book was published (int)
   * copiesAvailable: The number of copies available in the library (int)
2. **Constructor Overloading**: The Book class will have multiple constructors to initialize the objects in different ways:
   * A constructor that takes only the title and author.
   * A constructor that takes the title, author, and ISBN.
   * A constructor that takes all attributes: title, author, ISBN, yearPublished, and copiesAvailable.
3. **Methods**:
   * A method displayBookInfo() to display the details of the book.
   * A method updateCopies(int newCopies) to update the number of available copies.

**Task Description:** You are creating a Library Management System using a Book class that can be initialized in multiple ways through constructor overloading. You can create a book with just the title and author, or include the ISBN, or provide all details like year published and available copies. The system includes methods to display book details and update the number of available copies, allowing for flexible management of library books.

|  |
| --- |
|  |

**Task #3**

You are tasked with designing an **Online Course Management System** for a university. The system will handle information about various courses and instructors. You will need to implement classes that utilize **protected** and **private** variables, along with **constructor overloading** and method **overriding**.

1. **Classes**:
   * **Course Class**: This class should have the following attributes:
     + courseName (private): The name of the course.
     + courseID (private): A unique identifier for the course.
     + credits (protected): The number of credits the course offers.
   * **Instructor Class**: This class should inherit from the Course class and have the following attributes:
     + instructorName (private): The name of the instructor.
     + department (protected): The department the instructor belongs to.
2. **Constructor Overloading**:
   * The Course class should have multiple constructors:
     + One constructor that takes only courseName and courseID.
     + Another constructor that takes courseName, courseID, and credits.
   * The Instructor class should also have overloaded constructors:
     + One that takes only instructorName and department.
     + Another that takes instructorName, department, and an instance of Course to associate the instructor with a course.
3. **Method Overriding**:
   * Both classes should include a method called getDetails():
     + The Course class’s getDetails() method should return the course name and ID.
     + The Instructor class’s getDetails() method should override this to return the instructor’s name and associated course details.

**Task Description:** You are designing an Online Course Management System with two classes: Courseand Instructor**.**

* The **Course** class has private attributes for courseName and courseID, and a protected attribute for credits. It features overloaded constructors for different initialization options and a getDetails() method to display course information.
* The **Instructor** class inherits from the **Course** class, adding a private instructorName and a protected department. It also includes overloaded constructors and overrides the getDetails() method to show both instructor and associated course details.

The goal is to implement these classes in Java and create instances to verify the system's functionality.

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 5**

**Objective:**

The objective of this lab is to understand the concept of arrays (both single and multi-dimensional), array indexing, and how to traverse arrays using loops. Additionally, we aim to become familiar with the String class in Java and explore various operations such as comparison, extraction, and manipulation of strings.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

#### **Arrays**

An array in Java is a fixed-size, sequential collection of elements of the same type. Once an array is created, its size cannot be changed. The array elements are accessed using an index, which starts from 0.

**Syntax:**

elementType[] arrayRefVar = new elementType[arraySize]; // One-dimensional array

elementType[][] arrayRefVar; // Two-dimensional array

**To assign values:**

arrayRefVar[index] = value; // One-dimensional array

arrayRefVar[row][column] = value; // Two-dimensional array

#### **Strings in Java**

In Java, strings are represented as objects. The String class is immutable, which means once a string object is created, it cannot be modified.

**Syntax:**

String newString = new String("stringLiteral");

String newString = "stringLiteral";

#### **Common String Methods:**

1. equals(StringLiteral): Compares two strings for equality.
2. equalsIgnoreCase(StringLiteral): Compares strings ignoring case differences.
3. compareTo(StringLiteral): Compares two strings lexicographically.

**Procedure:**

#### **Calculate Average of an Array**

Write a program using a one-dimensional array to find the average of a set of numbers.

class Average {

public static void main(String args[]) {

double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};

double result = 0;

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

result += nums[i];

}

System.out.println("Average is " + result / nums.length);

}

}

#### **Initialize 2D Array with Random Numbers**

Write a program to initialize and print a 5x5 2D array with random numbers.

class RandomArray2D {

public static void main(String[] args) {

double[][] array2d = new double[5][5];

for (int row = 0; row < array2d.length; row++) {

for (int col = 0; col < array2d[row].length; col++) {

array2d[row][col] = Math.round(Math.random() \* 100);

}

}

for (int row = 0; row < array2d.length; row++) {

for (int col = 0; col < array2d[row].length; col++) {

System.out.print(array2d[row][col] + " ");

}

System.out.println();

}

}

}

#### **Explore String Methods**

Write a program to explore different methods of the String class, such as equals, equalsIgnoreCase, and getChars.

class GetCharsDemo {

public static void main(String args[]) {

String longStr = "This could have been a very long line...";

System.out.println(longStr);

String s = "This is a demo of the getChars method.";

int start = 10, end = 14;

char buf[] = new char[end - start];

s.getChars(start, end, buf, 0);

System.out.println(buf);

String s1 = "Hello";

String s2 = "Hello";

String s3 = "Goodbye";

String s4 = "HELLO";

System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));

System.out.println(s1 + " equals " + s3 + " -> " + s1.equals(s3));

System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " + s1.equalsIgnoreCase(s4));

}

}

|  |
| --- |
|  |

**Expected Outcome:**

**Task #1**

#### **Scenario:**

You are developing a program to help a weather monitoring system analyze temperature data for a city. The system collects daily temperature readings for 7 days, and you are required to generate a summary report based on this data.

#### **Task Description:**

Write a Java program that:

* Uses a one-dimensional array to store the temperature readings for 7 days.
* Calculates and displays the average temperature for the week.
* Identifies and prints the highest and lowest temperatures recorded.
* Traverses the array using a loop to print the temperature readings for each day.

|  |
| --- |
|  |

**Task #2**

#### **Scenario:**

You are working on an inventory management system for a retail store chain. The system keeps track of the stock levels for multiple products across several branches. The stock levels are stored in a 2D array, where each row represents a product and each column represents a branch.

#### **Task Description:**

Write a Java program that:

* Initializes a 2D array (with 5 products and 4 branches) with random stock levels between 0 and 100.
* Traverses the 2D array using nested loops to display the stock count for each product at each branch.
* Calculates the total stock of each product across all branches and identifies the product with the highest stock.

|  |
| --- |
|  |

**Task #3**

#### **Scenario:**

You are building a customer service application that needs to identify potential duplicate customer records in a list. The list contains customer names, and duplicates may appear with different capitalization (e.g., "John" and "john").

#### **Task Description:**

Write a Java program that:

* Uses a one-dimensional array to store a list of customer names.
* Compares the names in the array using equalsIgnoreCase() to check for duplicates (case-insensitive comparison).
* Prints the duplicate names found in the list.
* Traverses the array using loops to compare all possible pairs of customer names.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 6**

**Objective:**

The objective of this lab session is to understand the concept of inheritance, which includes learning about the superclass and subclass relationships in Java. We will explore how inheritance promotes code reuse and specialization in object-oriented programming by implementing various classes and methods.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

**Inheritance** is a fundamental feature of object-oriented programming that allows one class (subclass) to inherit the properties and behaviors (fields and methods) of another class (superclass). It enables the creation of hierarchical structures in which general attributes and methods are defined in a superclass and inherited by more specific subclasses. Key concepts include:

* **Superclass**: The class that is inherited from.
* **Subclass**: The class that inherits the properties and methods of the superclass.
* **Multilevel Inheritance**: A subclass can act as a superclass for another class, resulting in multiple levels of inheritance.
* **Method Overriding**: A subclass can provide its specific implementation of a method already defined in the superclass.

**Procedure:**

#### **Inheritance Basics**

This task demonstrates basic inheritance by creating a Box class and extending it with a BoxWeight class.

// This program uses inheritance to extend Box.

class Box {

double width;

double height;

double depth;

// construct clone of an object

Box(Box ob) {

width = ob.width;

height = ob.height;

depth = ob.depth;

}

// constructor used when all dimensions are specified

Box(double w, double h, double d) {

width = w;

height = h;

depth = d;

}

// default constructor when no dimensions are specified

Box() {

width = -1;

height = -1;

depth = -1;

}

// constructor used when a cube is created

Box(double len) {

width = height = depth = len;

}

// compute and return volume

double volume() {

return width \* height \* depth;

}

}

// Here, Box is extended to include weight.

class BoxWeight extends Box {

double weight; // weight of box

// constructor for BoxWeight

BoxWeight(double w, double h, double d, double m) {

width = w;

height = h;

depth = d;

weight = m;

}

}

class DemoBoxWeight {

public static void main(String args[]) {

BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);

BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);

double vol;

vol = mybox1.volume();

System.out.println("Volume of mybox1 is " + vol);

System.out.println("Weight of mybox1 is " + mybox1.weight);

System.out.println();

vol = mybox2.volume();

System.out.println("Volume of mybox2 is " + vol);

System.out.println("Weight of mybox2 is " + mybox2.weight);

}

}

##### Expected Output

Volume of mybox1 is 3000.0

Weight of mybox1 is 34.3

Volume of mybox2 is 24.0

Weight of mybox2 is 0.076

#### **Multilevel Inheritance**

This task demonstrates multilevel inheritance by extending BoxWeight with a Shipment class that adds a cost attribute.

// Add shipping costs.

class Shipment extends BoxWeight {

double cost;

// construct clone of an object

Shipment(Shipment ob) {

super(ob);

cost = ob.cost;

}

// constructor when all parameters are specified

Shipment(double w, double h, double d, double m, double c) {

super(w, h, d, m); // call superclass constructor

cost = c;

}

// default constructor

Shipment() {

super();

cost = -1;

}

// constructor used when a cube is created

Shipment(double len, double m, double c) {

super(len, m);

cost = c;

}

}

class DemoShipment {

public static void main(String args[]) {

Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);

Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.28);

double vol;

vol = shipment1.volume();

System.out.println("Volume of shipment1 is " + vol);

System.out.println("Weight of shipment1 is " + shipment1.weight);

System.out.println("Shipping cost: $" + shipment1.cost);

System.out.println();

vol = shipment2.volume();

System.out.println("Volume of shipment2 is " + vol);

System.out.println("Weight of shipment2 is " + shipment2.weight);

System.out.println("Shipping cost: $" + shipment2.cost);

}

}

##### Expected Output

swift

Copy code

Volume of shipment1 is 3000.0

Weight of shipment1 is 10.0

Shipping cost: $3.41

Volume of shipment2 is 24.0

Weight of shipment2 is 0.76

Shipping cost: $1.28

#### **Lab Assignment**

In this task, we will design a class hierarchy involving Person, Student, Employee, Faculty, and Staff.

class Person {

String name, address, phoneNumber, email;

Person(String name, String address, String phoneNumber, String email) {

this.name = name;

this.address = address;

this.phoneNumber = phoneNumber;

this.email = email;

}

public String toString() {

return "Person: " + name;

}

}

class Student extends Person {

final String status;

Student(String name, String address, String phoneNumber, String email, String status) {

super(name, address, phoneNumber, email);

this.status = status;

}

public String toString() {

return "Student: " + name;

}

}

class Employee extends Person {

String office;

double salary;

String dateHired;

Employee(String name, String address, String phoneNumber, String email, String office, double salary, String dateHired) {

super(name, address, phoneNumber, email);

this.office = office;

this.salary = salary;

this.dateHired = dateHired;

}

public String toString() {

return "Employee: " + name;

}

}

class Faculty extends Employee {

String officeHours, rank;

Faculty(String name, String address, String phoneNumber, String email, String office, double salary, String dateHired, String officeHours, String rank) {

super(name, address, phoneNumber, email, office, salary, dateHired);

this.officeHours = officeHours;

this.rank = rank;

}

public String toString() {

return "Faculty: " + name;

}

}

class Staff extends Employee {

String title;

Staff(String name, String address, String phoneNumber, String email, String office, double salary, String dateHired, String title) {

super(name, address, phoneNumber, email, office, salary, dateHired);

this.title = title;

}

public String toString() {

return "Staff: " + name;

}

}

public class TestClass {

public static void main(String[] args) {

Person person = new Person("John", "123 Main St", "555-1234", "john@example.com");

Student student = new Student("Alice", "456 Maple St", "555-5678", "alice@example.com", "Senior");

Employee employee = new Employee("Bob", "789 Oak St", "555-9876", "bob@example.com", "Office A", 60000, "01-01-2020");

Faculty faculty = new Faculty("Dr. Smith", "321 Birch St", "555-1357", "smith@example.com", "Office B", 80000, "05-01-2018", "9am-11am", "Professor");

Staff staff = new Staff("Mary", "654 Pine St", "555-2468", "mary@example.com", "Office C", 50000, "12-12-2021", "HR Manager");

System.out.println(person.toString());

System.out.println(student.toString());

System.out.println(employee.toString());

System.out.println(faculty.toString());

System.out.println(staff.toString());

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

You have been assigned the responsibility of designing a **Fleet Management System** for a transportation company that manages various vehicle types, specifically **Sedans** and **Cargo Trucks**. This system must leverage the principles of **inheritance**, **constructor overloading**, and **method overriding** to efficiently handle the specifications and operations of each vehicle type.

1. **Base Class - Vehicle**:
   * Create a base class named Vehicle, which includes the following attributes:
     + A private attribute manufacturer (String): indicating the manufacturer of the vehicle.
     + A private attribute model (String): representing the model of the vehicle.
     + A protected attribute manufacturingYear (int): indicating the year in which the vehicle was manufactured.
   * The Vehicle class should consist of:
     + A constructor that initializes the manufacturer, model, and manufacturingYear.
     + A method named displayDetails() that returns a formatted string showcasing the manufacturer, model, and manufacturing year of the vehicle.
2. **Derived Classes**:
   * **Sedan Class**:
     + Inherit from the Vehicle class.
     + Introduce a private attribute numberOfDoors (int): indicating how many doors the sedan has.
     + Provide a constructor that initializes the manufacturer, model, manufacturingYear, and numberOfDoors.
     + Override the displayDetails() method to include the number of doors in the output.
   * **CargoTruck Class**:
     + Inherit from the Vehicle class.
     + Add a private attribute cargoCapacity (double): representing the maximum weight the truck can carry.
     + Provide a constructor that initializes the manufacturer, model, manufacturingYear, and cargoCapacity.
     + Override the displayDetails() method to present the cargo capacity alongside the other vehicle details.

#### **Task Description:**

In the **Fleet Management System** for a transportation company, a base class named Vehicle is created with private attributes for the manufacturer and model, and a protected attribute for the manufacturing year. Two derived classes, Sedan and CargoTruck, inherit from the Vehicle class. The Sedan class includes an additional private attribute for the number of doors, while the CargoTruck class features a private attribute for cargo capacity. Each derived class has its own constructor to initialize all attributes and overrides the displayDetails() method to present specific information. This scenario showcases the principles of **inheritance**, **constructor overloading**, and **method overriding** in object-oriented programming.

|  |
| --- |
|  |

**Task #2**

#### **Scenario:**

An academic institution needs to manage the hierarchy of its employees, which includes general staff, faculty, and administrators. Each employee has a salary, but faculty members also have office hours and a rank. The system needs to keep track of these attributes and allow for customized printing of details based on the type of employee.

#### **Task Description:**

1. Create a base class Person with fields for name, address, phone number, and email address.
2. Extend Person to create an Employee class with fields for salary and the date of hiring.
3. Further extend Employee to create a Faculty class that adds fields for office hours and rank.
4. Override the toString() method in each class to output customized details for each type of person.
5. Write a main program to create objects for a generic person, employee, and faculty member, then print their details using the overridden toString() methods.

|  |
| --- |
|  |

**Task #3**

#### **Scenario:**

A company needs to track shipments of products, which have multiple layers of attributes such as size, weight, and shipping costs. The system must calculate the volume, weight, and total cost of shipping a product based on its attributes.

#### **Task Description:**

1. Create a Box class with width, height, and depth, along with a method to calculate the volume.
2. Extend Box to a BoxWeight class, which adds a weight attribute.
3. Extend BoxWeight into a Shipment class, which adds a cost attribute for shipping.
4. Write a main program that creates shipment objects, calculates and displays their volume, weight, and shipping cost.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 7**

**Objective:**

The objective of this lab session is to understand the concepts of abstract methods and classes, as well as final methods and classes in Java. We will explore how these concepts facilitate a structured approach to object-oriented programming by implementing various classes and methods and Open Ended Lab

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

Inheritance is a key principle of object-oriented programming that allows one class (subclass) to inherit properties and behaviors (fields and methods) from another class (superclass). This facilitates the creation of hierarchical structures, enabling general attributes and methods to be defined in a superclass and inherited by more specific subclasses. Key concepts include:

* **Abstract Methods**: Methods that must be implemented by subclasses.
* **Abstract Classes**: Classes that contain abstract methods and cannot be instantiated.
* **Final Methods**: Methods that cannot be overridden by subclasses.
* **Final Classes**: Classes that cannot be inherited from.

**Procedure:**

### **1. Abstract Class and Methods**

1. **Define an Abstract Class**:
   * Create an abstract class Account with attributes for account ID and balance, and abstract methods for withdrawal and deposit.

abstract class Account {

protected String id;

protected double balance;

public Account(String id, double balance) {

this.id = id;

this.balance = balance;

}

public String getID() {

return id;

}

public double getBalance() {

return balance;

}

public abstract boolean withdraw(double amount);

public abstract void deposit(double amount);

}

1. **Implement a Subclass**:
   * Create a SavingsAccount class that extends Account and implements the withdrawal and deposit methods with the specified conditions.

class SavingsAccount extends Account {

public SavingsAccount(String id, double initialDeposit) {

super(id, initialDeposit >= 10 ? initialDeposit : throw new IllegalArgumentException("Initial deposit must be at least $10"));

}

@Override

public void deposit(double amount) {

balance += amount;

}

@Override

public boolean withdraw(double amount) {

if (balance - amount - 2 < 10) {

return false; // Insufficient funds after withdrawal fee

}

balance -= amount + 2; // Deduct the transaction fee

return true; // Withdrawal successful

}

}

1. **Test the Implementation**:
   * Create a main method to test the SavingsAccount functionality.

public class BankApplication {

public static void main(String[] args) {

SavingsAccount account = new SavingsAccount("12345", 50.0);

account.deposit(20);

System.out.println("Balance after deposit: " + account.getBalance());

boolean result = account.withdraw(30);

System.out.println("Withdrawal successful: " + result);

System.out.println("Balance after withdrawal: " + account.getBalance());

}

}

### **Final Methods and Classes**

1. **Define a Final Class**:
   * Create a class marked as final to prevent inheritance.

final class FinalClass {

public final void displayMessage() {

System.out.println("This is a final method in a final class.");

}

}

1. **Test Final Class**:
   * Attempt to extend the final class to demonstrate that it cannot be inherited.

class AttemptInheritance extends FinalClass { // This will cause a compile-time error

public void newMethod() {

System.out.println("Trying to inherit.");

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

### **Scenario:**

A banking application needs to manage different types of accounts, including savings and checking accounts. The system should ensure that all accounts can perform common operations such as deposits and withdrawals, while also allowing for specific implementations based on the type of account.

### Task Description:

1. **Create an Abstract Class**:
   * Define an abstract class named Account with the following attributes:
     + protected String id
     + protected double balance
   * Implement the following methods:
     + public Account(String id, double balance): A constructor that initializes the account ID and balance.
     + public String getID(): Returns the account ID.
     + public double getBalance(): Returns the current balance.
     + public abstract boolean withdraw(double amount): An abstract method for withdrawing money.
     + public abstract void deposit(double amount): An abstract method for depositing money.
2. **Create Subclass**:
   * Implement a subclass named SavingsAccount that extends Account and includes:
     + A constructor that requires an initial deposit of at least $10.
     + Implementation of the withdraw method, including a transaction fee of $2 for each withdrawal, ensuring that the balance does not drop below $10 after withdrawal.
3. **Test the Implementation**:
   * In the main method, create an instance of SavingsAccount, perform some deposits and withdrawals, and print the resulting balances to verify the functionality.

|  |
| --- |
|  |

**Task #2**

### **Scenario:**

A logistics company wants to manage the shipping of products with strict guidelines. Certain types of boxes should not be altered or inherited further, as they represent standardized shipping containers.

### **Task Description:**

1. **Create a Final Class**:
   * Define a class named ShippingBox as a final class that includes:
     + double width, double height, double depth: Attributes for the dimensions of the box.
     + A constructor that initializes these dimensions.
     + A method double calculateVolume(): Calculates and returns the volume of the box.
2. **Prevent Inheritance**:
   * Ensure that ShippingBox cannot be inherited by any other class.
3. **Demonstrate Functionality**:
   * Write a main method that creates an instance of ShippingBox, calculates its volume, and displays the result. Attempting to extend ShippingBox in another class should result in a compile-time error.

|  |
| --- |
|  |

**Task #3**

### **Scenario:**

An educational institution needs to manage various staff roles, including faculty and administrative personnel. It requires a system that distinguishes between different types of employees while enforcing specific rules about roles and responsibilities.

### **Task Description:**

1. **Create a Base Class**:
   * Define a class named Person with fields for name, address, phone number, and email. Implement a constructor to initialize these fields.
2. **Use Abstract Classes**:
   * Extend Person to create an abstract class named Employee that includes:
     + Fields for salary and date hired.
     + An abstract method public abstract String getDetails(): To retrieve employee details.
3. **Implement Subclasses**:
   * Create two subclasses: Faculty and Staff, which inherit from Employee and implement the getDetails() method to return customized details specific to their roles.
4. **Test the Implementation**:
   * In the main method, create instances of Faculty and Staff, print their details using the getDetails() method to demonstrate the use of inheritance and polymorphism.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 7 (Open-Ended Lab)**

|  | **Open Ended Lab** |
| --- | --- |
| **Bloom's Taxonomy** | **GAs** |
| A3  C3  C5 | GA-2  GA-4  GA-6 |

**Title:** Designing a Hospital Management System using Object-Oriented Programming Principles

**Motivation:**

The purpose of this Open-Ended Lab (OEL) is to give students practical experience in applying Object-Oriented Programming (OOP) principles to develop a functional hospital management system. By focusing on real-world scenarios, students will learn to implement key OOP concepts like Encapsulation, Inheritance, Polymorphism, and Abstraction in developing a robust system to handle various hospital operations.

**Concept (Problem Statement):**

In this lab, students will design and implement a Hospital Management System using Java. This system should manage the relationships between hospitals, patients, doctors, wards, and staff. The system must use OOP principles to create a flexible and efficient solution to handle the interactions between these entities.

**Design:**

* **Classes and Objects:** Create classes for Hospital, Ward, Person, Patient, Doctor, and Staff. Each class should encapsulate the necessary data and provide methods for interacting with that data. For example, the Doctor class can have methods to manage patient assignments and ward duties.
* **Inheritance:** Implement a hierarchy of people where Patient, Doctor, and Staff inherit from a base Person class. This will allow for the addition of future roles within the hospital system without rewriting core functionality.
* **Polymorphism:** Use polymorphism to handle the relationship between different types of people. For instance, a method to assign a person to a ward can work for both Patient and Doctor by using a common interface.
* **Abstraction:** Create abstract classes or interfaces such as MedicalEntity or Assignable to ensure consistency in assigning people to different hospital facilities. This can include functionality to handle interactions between entities such as WardAssignment or MedicalTreatment.

**Key Functionalities:**

* **Patient Management:** Implement methods to add, view, and update patient records. These records should track personal details, medical history, current ward, and assigned doctors.
* **Ward Management:** Enable hospitals to manage wards and their capacity. The system should allow for ward assignments based on bed availability and specific medical needs.
* **Doctor and Staff Assignments:** Implement functionality to manage doctor and staff assignments to specific wards or patients. This should include tracking shifts, availability, and specialization.
* **Hospital Reporting:** Generate reports to summarize hospital operations such as patient occupancy per ward, doctor assignments, and patient treatment statuses.

**Deliverables**

**Background/Theory:**

**Procedure / Methodology:**

**Flowchart / Block diagram:**

**Analysis:**

**Results:**

**Discussion on Results:**

**Concluding Remarks:**

**Reference:**

**Lab Session 8**

**Objective:**

The objective of these lab sessions is to understand the concepts of packages, interfaces, The labs will demonstrate how these features enhance object-oriented programming, code structure, and runtime error management.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Lab Session A: Packages and Interfaces**

### **Objective:**

Understand the usage of packages and interfaces in Java to facilitate better organization and multiple inheritance.

**Introduction:**

In Java, **packages** help manage large codebases by organizing classes into namespaces, preventing name conflicts. **Interfaces** provide a way to specify a set of methods that one or more classes must implement, enabling multiple inheritance since a class can implement multiple interfaces.

**Procedure:**

#### 1. **Packages in Java**

// A simple package example

// File: MyPack/AccountBalance.java

package MyPack;

class Balance {

String name;

double bal;

Balance(String n, double b) {

name = n;

bal = b;

}

void show() {

if (bal < 0)

System.out.print("--> ");

System.out.println(name + ": $" + bal);

}

}

class AccountBalance {

public static void main(String args[]) {

Balance current[] = new Balance[3];

current[0] = new Balance("K. J. Fielding", 123.23);

current[1] = new Balance("Will Tell", 157.02);

current[2] = new Balance("Tom Jackson", -12.33);

for (int i = 0; i < 3; i++)

current[i].show();

}

}

#### **Steps:**

1. Save the file under a folder named MyPack.
2. Compile it:

MyPack/AccountBalance.java

1. Execute the program:

MyPack.AccountBalance

#### 2. **Interface Implementation in Java**

// Demonstrating multiple inheritance using interfaces

interface Crawlable {

void crawl();

}

interface Moveable {

void move();

}

class Animal implements Crawlable, Moveable {

public void crawl() {

System.out.println("The animal is crawling.");

}

public void move() {

System.out.println("The animal is moving.");

}

}

public class InterfaceDemo {

public static void main(String[] args) {

Animal animal = new Animal();

animal.crawl();

animal.move();

}

}

#### **Steps:**

1. Implement the interfaces Crawlable and Moveable in the Animal class.
2. Run the InterfaceDemo class to test the multiple inheritance.

#### **3.** CharSequence Interface Implementation

// Implementing CharSequence interface to reverse a string

class ReverseString implements CharSequence {

private String data;

public ReverseString(String data) {

this.data = data;

}

public int length() {

return data.length();

}

public char charAt(int index) {

return data.charAt(length() - index - 1);

}

public CharSequence subSequence(int start, int end) {

return new ReverseString(data.substring(start, end));

}

public String toString() {

return new StringBuilder(data).reverse().toString();

}

public static void main(String[] args) {

ReverseString reverse = new ReverseString("Hello, Java!");

System.out.println("Reversed String: " + reverse.toString());

}

}

#### **Steps:**

1. Implement the CharSequence interface and reverse a string.
2. Test the implementation using the main method.

**Expected Outcome:**

|  |
| --- |
|  |

**Lab Session B: Exception Handling**

### **Objective:**

Understand how runtime errors are handled in Java using exception handling mechanisms like try, catch, throw, throws, and finally.

**Introduction:**

In Java, exceptions allow the program to handle runtime errors gracefully. The try-catch block is used to catch and process exceptions, while the finally block ensures the execution of certain statements regardless of whether an exception was thrown. This makes programs robust and error-tolerant.

### 1. try

* The try block contains code that might throw an exception.
* It is used to define a block of code to be tested for exceptions while the program is running.
* If an exception occurs in the try block, the flow of control moves to the corresponding catch block.

### 2. catch

* The catch block is used to handle the exception that occurs in the try block.
* Each catch block must specify the type of exception it can handle (like ArithmeticException, NullPointerException, etc.).

### 3. finally

* The finally block contains code that is always executed after the try and catch blocks, regardless of whether an exception was thrown or not.
* It's typically used to clean up resources like closing a file, closing a database connection, etc.

### . throw

* The throw keyword is used to explicitly throw an exception.
* It can be used to throw either a checked or unchecked exception.
* It is commonly used to throw custom exceptions or re-throw existing exceptions.

### 5. throws

* The throws keyword is used in the method signature to declare exceptions that a method might throw.
* It is used for checked exceptions to tell the caller of the method that they must handle the exception or propagate it further.

### Checked and Unchecked Exceptions

* Checked Exceptions: These are exceptions that are checked at compile time. If a method might throw a checked exception, it must either handle it using a try-catch block or declare it using the throws keyword (e.g., IOException).
* Unchecked Exceptions: These are exceptions that are not checked at compile time. They typically result from programming errors (e.g., NullPointerException, ArithmeticException).

**Procedure:**

#### 1. **Basic Calculator with Exception Handling with** **ArithmeticException:**

public class ArithmeticExceptionExample {

public static void main(String[] args) {

int numerator = 10;

int denominator = 0;

try {

// Attempt to divide by zero, which will throw ArithmeticException

int result = numerator / denominator;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero is not allowed.");

System.out.println("Exception message: " + e.getMessage());

}

}

}

#### **Steps:**

1. Enter non-numeric input to trigger the exception.
2. Verify that the exception is handled, and the program terminates gracefully.

#### **NullPointerExceptionExample** **Demonstration:**

public class NullPointerExceptionExample {

public static void main(String[] args) {

// Create a String reference and set it to null

String str = null;

try {

// Attempt to call a method on a null reference

System.out.println("String length: " + str.length()); // This will throw NullPointerException

} catch (NullPointerException e) {

System.out.println("Error: Tried to perform an operation on a null object.");

System.out.println("Exception message: " + e.getMessage());

}

}

}

#### 3. **IndexOutOfBoundsException Demonstration**

public class IndexOutOfBoundsExceptionArrayExample {

public static void main(String[] args) {

// Create an array with three elements

int[] numbers = {10, 20, 30};

try {

// Attempt to access an invalid index (e.g., index 5)

System.out.println("Accessing element at index 5: " + numbers[5]); // This will throw IndexOutOfBoundsException

} catch (IndexOutOfBoundsException e) {

System.out.println("Error: Tried to access an invalid index in the array.");

System.out.println("Exception message: " + e.getMessage());

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

**Scenario:** You are tasked with developing a banking application that performs various operations like deposits, withdrawals, and balance inquiries. The application needs to handle invalid input and runtime errors effectively, such as attempting to withdraw more money than available.

### Task Description:

1. **Create a class BankAccount** with methods for deposit(double amount) and withdraw(double amount).
2. **Implement exception handling** to manage:
   * IllegalArgumentException for negative deposit/withdrawal amounts.
   * InsufficientFundsException for withdrawals that exceed the current balance.
3. **Write a main method** to simulate user interactions, capturing any exceptions thrown and displaying appropriate error messages.

|  |
| --- |
|  |

**Task #2**

A software company is developing an **Online Payment System** that supports multiple payment methods like **Credit Card**, **PayPal**, and **Bank Transfer**. They want to ensure that the system handles payment failures smoothly, allowing the user to retry or select a different method without the program crashing.

The company has created an interface called PaymentMethod, which includes a method processPayment(double amount) that each payment class (CreditCard, PayPal, BankTransfer) must implement. During the payment process, certain exceptions can occur, such as:

1. **InvalidCardException**: Raised if the user inputs an invalid credit card number.
2. **InsufficientFundsException**: Raised if the user's bank account or PayPal balance has insufficient funds.
3. **PaymentNetworkException**: Raised if there is a network issue while processing the payment.

**Your task:**

1. **Define the PaymentMeth**BankTransfer—that implement PaymentMethod and throw the appropriate exception**od interface** with the method processPayment(double amount) that throws the potential exceptions.
2. **Create three classes**—CreditCard, PayPal, and s based on the following scenarios:
   * CreditCard: Throws an InvalidCardException if the card number is invalid.
   * PayPal: Throws an InsufficientFundsException if the PayPal balance is too low.
   * BankTransfer: Throws a PaymentNetworkException if a network issue occurs.
3. In the main class, **simulate the payment process** by allowing the user to choose a payment method and handle the exceptions gracefully using try-catch blocks, prompting the user to retry if an exception is thrown.

### **Task Description:**

The task involves developing an Online Payment System with multiple payment methods: Credit Card, PayPal, and Bank Transfer. Each payment method implements a common PaymentMethod interface with the method processPayment(double amount). During the payment process, specific exceptions can occur:

* InvalidCardException for invalid credit card numbers.
* InsufficientFundsException for insufficient funds in PayPal or bank accounts.
* PaymentNetworkException for network issues during a bank transfer.

The system must handle these exceptions gracefully, allowing users to retry or choose a different payment method without crashing. The main class will simulate the payment process and manage errors using try-catch blocks.

|  |
| --- |
|  |

**Task #3**

**Scenario:** In a transportation system, you need to manage various modes of transport like cars, bikes, and trucks. Each mode of transport has specific capabilities but should also adhere to common transportation functionalities.

### Task Description:

1. Create a package named Transport to contain all transport-related classes.
2. Define two interfaces**:** Drivable with methods such as drive() and stop(), and Loadable with methods such as loadCargo(int weight) and unloadCargo().
3. Implement classes Car**,** Bike, andTruck that implement the Drivable interface. The Truck class should also implement the Loadable interface. Each class should define its unique attributes and methods while adhering to the interface contracts.
4. Create main class in package mainpkg and implement all classes in main class.
5. Show the result.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 9**

**Objective:**

To comprehensively explore and implement various GUI components, layouts, and charts in JavaFX, enabling the creation of interactive and visually appealing applications, including a user-friendly login page that demonstrates effective user input handling and data representation within the application. This integrated approach aims to enhance understanding of JavaFX's capabilities and improve skills in developing modern Java applications.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

JavaFX is the recommended toolkit for building GUIs in Java, providing a more modern approach compared to Swing. It allows the creation of rich graphical applications with various controls, layouts, and the integration of multimedia. This lab will cover the basic components of JavaFX and demonstrate how to utilize them effectively in application development.

**Procedure:**

1. **Creating a JavaFX Application:**
   * Import the necessary JavaFX packages:

import javafx.application.Application;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.stage.Stage;

1. **Defining the Main Class:**
   * Create a class that extends Application:

public class JavaFxControls extends Application {

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage stage) {

// GUI components will be defined here

}

}

1. **Creating GUI Components:**
   * Add various controls like Labels, Buttons, RadioButtons, CheckBoxes, ComboBoxes, and DatePickers:

Label heading = new Label("JavaFX Controls");

heading.setFont(Font.font("Verdana", 30));

Button button1 = new Button("Wrong");

Button button2 = new Button("Accept");

1. **Using Layout Managers:**
   * Utilize HBox and VBox to arrange components:

HBox h = new HBox(20);

h.getChildren().addAll(button1, button2);

VBox v = new VBox(20);

v.setPadding(new Insets(20));

v.getChildren().addAll(heading, h);

1. **Setting Up the Scene:**
   * Create a scene and set it to the stage:

Scene scene = new Scene(v, 500, 500);

stage.setScene(scene);

stage.show();

**Expected Outcomes:**

|  |
| --- |
|  |

**Task #1**

**Scenario:** You are tasked with creating a simple application that collects user information.

**Task Description:**

1. Create a JavaFX application that includes:
   * A label for the title.
   * Two buttons ("Submit" and "Cancel").
   * A text field for user input.
   * A combo box for selecting a city.
   * A date picker for selecting a date.
2. Arrange the components using VBox and HBox.

|  |
| --- |
|  |

**Task #2**

**Scenario:** Enhance the previous application by adding more interactivity.

**Task Description:**

1. Implement event handling for the buttons:
   * The "Submit" button should print the collected information to the console.
   * The "Cancel" button should clear all input fields.
2. Use CheckBox to allow users to select their preferred mode of communication (Email, Phone, etc.).

|  |
| --- |
|  |

**Task #3**

**Scenario:** Develop a JavaFX application that simulates a simple user registration and profile management system.

**Task Description:**

1. **Create Two Scenes:**
   * **Registration Scene:**
     + Include components for user input such as:
       - Labels for "Username", "Password", and "Email".
       - Text fields for entering the username and email.
       - A password field for entering the password.
       - A "Register" button to submit the information.
     + Use a VBox layout to arrange these components vertically.
   * **Profile Scene:**
     + Include components to display user information:
       - A label to show the registered username.
       - A label for the registered email.
       - A "Back to Registration" button to return to the registration scene.
     + Use a VBox layout to arrange these components vertically.
2. **Implement Navigation:**
   * Set up event handling for the "Register" button:
     + When clicked, validate the user input (ensure fields are not empty).
     + If valid, switch to the Profile Scene and display the entered information.
   * Implement the "Back to Registration" button to return to the registration scene without losing the entered information.
3. **Add Validation:**
   * Display error messages in a Label if any of the input fields are empty when the "Register" button is clicked.
4. **Enhance User Experience:**
   * Style the application using CSS to make it visually appealing.
   * Add some transitions or animations when switching between scenes to enhance user experience.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 10**

**Objective:**

To understand and implement different layouts and charts in JavaFX to enhance user interfaces.

**Introduction:**

JavaFX provides various layout managers that help in organizing and arranging GUI components effectively. These layouts dictate how components are displayed and interact with each other within a scene. Below are some of the key layout classes and their descriptions:

|  |  |
| --- | --- |
| **Class** | **Description** |
| **BorderPane** | Organizes nodes into top, left, right, center, and bottom areas. |
| **FlowPane** | Arranges nodes in horizontal rows, wrapping to the next line if space is insufficient. |
| **GridPane** | Organizes nodes into a grid of rows and columns. |
| **HBox** | Aligns nodes in a single horizontal row. |
| **VBox** | Aligns nodes in a single vertical column. |
| **StackPane** | Stacks nodes on top of each other. |

JavaFX also supports various types of charts that visually represent data, such as pie charts, bar charts, and line charts. The charts are defined in the javafx.scene.chart package.

**Procedure:**

#### **1. FlowPane Example**

Create a simple application using FlowPane to arrange buttons horizontally.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.FlowPane;

import javafx.stage.Stage;

public class FlowPaneExample extends Application {

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("FlowPane Example");

Button button1 = new Button("Button 1");

Button button2 = new Button("Button 2");

Button button3 = new Button("Button 3");

FlowPane flowPane = new FlowPane();

flowPane.getChildren().addAll(button1, button2, button3);

Scene scene = new Scene(flowPane, 300, 100);

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Steps:**

1. Create a JavaFX project in your IDE.
2. Implement the above code in a class named FlowPaneExample.
3. Run the application to observe how the buttons are arranged.

#### **2. BorderPane Example**

Create an application that uses BorderPane to display components in various sections.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.scene.layout.BorderPane;

import javafx.stage.Stage;

public class BorderPaneExample extends Application {

@Override

public void start(Stage primaryStage) {

BorderPane borderPane = new BorderPane();

borderPane.setTop(new Label("Top Label"));

borderPane.setLeft(new Label("Left Label"));

borderPane.setRight(new Label("Right Label"));

borderPane.setCenter(new Label("Center Label"));

borderPane.setBottom(new Label("Bottom Label"));

Scene scene = new Scene(borderPane, 600, 400);

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Steps:**

1. Implement the above code in a class named BorderPaneExample.
2. Run the application to see how the labels are organized in different sections.

#### **3. GridPane Example**

Create an application using GridPane to arrange buttons in a grid layout.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

public class GridPaneExample extends Application {

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("GridPane Example");

Button button1 = new Button("Button 1");

Button button2 = new Button("Button 2");

Button button3 = new Button("Button 3");

Button button4 = new Button("Button 4");

GridPane gridPane = new GridPane();

gridPane.add(button1, 0, 0); // column 0, row 0

gridPane.add(button2, 1, 0); // column 1, row 0

gridPane.add(button3, 0, 1); // column 0, row 1

gridPane.add(button4, 1, 1); // column 1, row 1

Scene scene = new Scene(gridPane, 200, 100);

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Steps:**

1. Implement the above code in a class named GridPaneExample.
2. Run the application to see how the buttons are organized in a grid.

#### **4. Pie Chart Example**

Create a simple application to display a pie chart using JavaFX.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.chart.PieChart;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

public class PieChartExample extends Application {

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Pie Chart Example");

PieChart pieChart = new PieChart();

PieChart.Data slice1 = new PieChart.Data("Category A", 30);

PieChart.Data slice2 = new PieChart.Data("Category B", 70);

pieChart.getData().addAll(slice1, slice2);

VBox vbox = new VBox(pieChart);

Scene scene = new Scene(vbox, 400, 300);

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Steps:**

1. Implement the above code in a class named PieChartExample.
2. Run the application to see how the pie chart displays data.

**Expected Outcomes:**

|  |
| --- |
|  |

**Task #1**

#### **Scenario:** Building a Dashboard Application

**Context:** You are tasked with creating a dashboard application for a small business. The dashboard needs to display key performance indicators (KPIs) in a structured manner, including sales data, user engagement metrics, and recent activities.

**Task Description:**

1. **Design a BorderPane Layout**: Create a layout using BorderPane where:
   * The top section displays the title "Business Dashboard".
   * The left section shows a list of KPIs (e.g., Total Sales, Active Users).
   * The right section contains a pie chart representing the distribution of sales across different product categories.
   * The center section includes a grid of buttons that navigate to different functionalities (e.g., View Reports, Manage Users).
   * The bottom section has a footer with the message "© 2024 Business Inc.".

|  |
| --- |
|  |

**Task #2**

#### **Scenario:** Creating a Photo Gallery

**Context:** You are developing a photo gallery application that allows users to browse images and view their details. The application needs to display images in a responsive way.

**Task Description:**  
2. **Implement a FlowPane Layout**: Create a FlowPane that:

* Displays a collection of images as buttons (e.g., each button shows a thumbnail).
* Ensures that the images wrap to the next line if the screen width is reduced.
* Include a feature where clicking an image button opens a new window displaying the full image along with its title and description.

|  |
| --- |
|  |

**Task #3**

#### **Scenario:** Educational Quiz Application

**Context:** You are creating an educational quiz application that presents questions to users in a visually engaging manner. The application should also show the performance of users over time.

**Task Description:**  
3. **Utilize a GridPane for Quiz Questions and a Bar Chart for Performance**:

* Use a GridPane to display multiple choice questions, where each question is in its own row and each option is presented as a button within a column.
* After the user submits their answers, display a bar chart that shows the number of correct answers versus incorrect answers for the quiz, using a BarChart for visualization.
* Ensure that the layout is responsive and maintains a clear structure even when there are multiple questions and options.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 11**

**Objective:**

To Design a Login Page Using JavaFX Components

**Introduction:**

#### Login Window Overview:

To create a login form, two class files are utilized:

* **NextPage.java**
* **Login.java**

In **Login.java**, we set up two text fields for the username and password. A button is created to perform actions. The method text1.getText() retrieves the username, and text2.getText() retrieves the password entered by the user. If the values match "admin" for the username and "password" for the password, the user is directed to the next page upon clicking the submit button. The **NextPage.java** class is responsible for transitioning the user to the next page. If the user enters invalid credentials, an error message is displayed.

**Procedure:**

#### **Login.java**

This class handles the login form, validates user inputs, and navigates to the next page.

import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.scene.effect.DropShadow;

import javafx.scene.effect.Reflection;

import javafx.scene.layout.BorderPane;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.HBox;

import javafx.scene.paint.Color;

import javafx.scene.text.Font;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class Login extends Application {

String user = "admin";

String pw = "password";

String checkUser, checkPw;

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Login Form");

BorderPane bp = new BorderPane();

bp.setPadding(new Insets(10, 50, 50, 50));

// Adding HBox

HBox hb = new HBox();

hb.setPadding(new Insets(20, 20, 20, 30));

// Adding GridPane

GridPane gridPane = new GridPane();

gridPane.setPadding(new Insets(20, 20, 20, 20));

gridPane.setHgap(5);

gridPane.setVgap(5);

// Implementing Nodes for GridPane

Label lblUserName = new Label("Username");

final TextField txtUserName = new TextField();

Label lblPassword = new Label("Password");

final PasswordField pf = new PasswordField();

Button btnLogin = new Button("Login");

final Label lblMessage = new Label();

// Adding Nodes to GridPane layout

gridPane.add(lblUserName, 0, 0);

gridPane.add(txtUserName, 1, 0);

gridPane.add(lblPassword, 0, 1);

gridPane.add(pf, 1, 1);

gridPane.add(btnLogin, 2, 1);

gridPane.add(lblMessage, 1, 2);

// Reflection for gridPane

Reflection r = new Reflection();

r.setFraction(0.7f);

gridPane.setEffect(r);

// DropShadow effect

DropShadow dropShadow = new DropShadow();

dropShadow.setOffsetX(5);

dropShadow.setOffsetY(5);

// Adding text and DropShadow effect to it

Text text = new Text("Login Form");

text.setFont(Font.font("Verdana", 30));

text.setEffect(dropShadow);

// Adding text to HBox

hb.getChildren().add(text);

// Action for btnLogin

btnLogin.setOnAction(event -> {

checkUser = txtUserName.getText().trim();

checkPw = pf.getText().trim();

if (checkUser.isEmpty() || checkPw.isEmpty()) {

lblMessage.setText("Username or password cannot be empty.");

lblMessage.setTextFill(Color.RED);

} else if (checkUser.equals(user) && checkPw.equals(pw)) {

lblMessage.setText("Congratulations! Redirecting...");

lblMessage.setTextFill(Color.GREEN);

new NextPage(); // Create an instance of NextPage

primaryStage.close(); // Close login stage

} else {

lblMessage.setText("Incorrect username or password.");

lblMessage.setTextFill(Color.RED);

}

txtUserName.clear();

pf.clear();

});

// Add HBox and GridPane layout to BorderPane Layout

bp.setTop(hb);

bp.setCenter(gridPane);

// Adding BorderPane to the scene and loading CSS

Scene scene = new Scene(bp);

primaryStage.setScene(scene);

primaryStage.setResizable(false);

primaryStage.show();

}

}

#### **NextPage.java**

#### This class creates a new window to add user details, featuring fields for name, email, gender, education, location, and date of birth.

import java.time.LocalDate;

import javafx.application.Application;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.DatePicker;

import javafx.scene.control.Label;

import javafx.scene.control.ListView;

import javafx.scene.control.RadioButton;

import javafx.scene.control.TextField;

import javafx.scene.control.ToggleGroup;

import javafx.scene.layout.GridPane;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class NextPage extends Application {

Stage window;

public NextPage() {

window = new Stage();

window.setTitle("Add User");

window.setHeight(370);

window.setWidth(400);

window.setResizable(false);

addComponents();

window.show();

}

private void addComponents() {

Label name = new Label("Name");

TextField ntext = new TextField();

Label email = new Label("Email");

TextField etext = new TextField();

Label gender = new Label("Gender");

ToggleGroup group = new ToggleGroup();

RadioButton rmale = new RadioButton("Male");

RadioButton rfemale = new RadioButton("Female");

rmale.setToggleGroup(group);

rfemale.setToggleGroup(group);

Label edu = new Label("Education");

ObservableList<String> items = FXCollections.observableArrayList(

"PhD", "Master", "Graduate", "Intermediate", "Matric");

ListView<String> eduList = new ListView<>(items);

eduList.setPrefHeight(40);

Label loc = new Label("Location");

ComboBox<String> locList = new ComboBox<>();

locList.getItems().addAll("Karachi", "Islamabad", "Multan", "Lahore", "Peshawar");

Label dob = new Label("DOB");

DatePicker date = new DatePicker();

date.setValue(LocalDate.now());

Button btnSignup = new Button("Add User");

Button btnClear = new Button("Clear");

GridPane layout = new GridPane();

layout.setPadding(new Insets(20));

layout.setVgap(10);

layout.add(name, 0, 1);

layout.add(ntext, 1, 1);

layout.add(email, 0, 2);

layout.add(etext, 1, 2);

layout.add(gender, 0, 3);

layout.add(rmale, 1, 3);

layout.add(rfemale, 1, 3);

layout.setMargin(rfemale, new Insets(0, 0, 0, 80));

layout.add(edu, 0, 4);

layout.add(eduList, 1, 4);

layout.add(loc, 0, 5);

layout.add(locList, 1, 5);

layout.add(dob, 0, 6);

layout.add(date, 1, 6);

layout.add(btnSignup, 1, 7);

layout.add(btnClear, 1, 7);

layout.setMargin(btnClear, new Insets(0, 0, 0, 80));

btnSignup.setOnAction(event -> {

Alert alert = new Alert(AlertType.INFORMATION);

alert.setHeaderText(null);

alert.setContentText("Added successfully!");

alert.show();

});

Scene scene = new Scene(layout);

window.setScene(scene);

}

public static void main(String[] args) {

launch(args);

}

}

#### **Menus Example**

To further enhance your application, you can add a menu bar that provides various functionalities:

import javafx.application.Application;

import javafx.application.Platform;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.BorderPane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

public class MenuTest extends Application {

@Override

public void start(Stage primaryStage) {

BorderPane root = new BorderPane();

Scene scene = new Scene(root, 300, 250, Color.WHITE);

MenuBar menuBar = new MenuBar();

menuBar.prefWidthProperty().bind(primaryStage.widthProperty());

root.setTop(menuBar);

// File menu - new, save, exit

Menu fileMenu = new Menu("File");

MenuItem newMenuItem = new MenuItem("New");

MenuItem saveMenuItem = new MenuItem("Save");

MenuItem exitMenuItem = new MenuItem("Exit");

exitMenuItem.setOnAction(actionEvent -> Platform.exit());

fileMenu.getItems().addAll(newMenuItem, saveMenuItem, new SeparatorMenuItem(), exitMenuItem);

// Adding menus to the menu bar

menuBar.getMenus().addAll(fileMenu);

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Expected Outcomes:**

|  |
| --- |
|  |

**Task #1**

**Scenario:** You are developing a JavaFX application that requires user authentication to access certain features. The login page should allow users to enter their username and password. If the credentials are valid, the user is redirected to the main application interface; otherwise, an error message is displayed.

**Task Description:**

1. Create a JavaFX application with a login page that includes two text fields: one for the username and one for the password.
2. Implement a button that validates the entered credentials against a predefined username and password (e.g., "admin" and "password").
3. Display an appropriate message indicating whether the login was successful or if the credentials were incorrect, and ensure that the input fields are cleared after each attempt.

|  |
| --- |
|  |

**Task #2**

**Scenario:** Users often forget their passwords or make errors when entering their usernames. To improve the user experience, your application should provide real-time feedback when the user enters invalid data and allow them to reset their password if they forget it.

**Task Description:**

1. Modify the existing login page to include input validation that checks if the username or password fields are empty, displaying an error message immediately if so.
2. Add a "Forgot Password?" link that, when clicked, opens a dialog prompting the user to enter their registered email address for password recovery.
3. Implement a function that simulates sending a password reset email and displays a success message once the action is triggered.

|  |
| --- |
|  |

**Task #3**

**Scenario:** Your application now requires a user registration feature, allowing new users to create an account. This new functionality should be accessible from the login page and should collect essential user information, such as username, password, and email.

**Task Description:**

1. Add a "Register" button to the login page that, when clicked, opens a new window containing a registration form.
2. The registration form should include fields for entering a username, password, and email address, with appropriate validation for each (e.g., checking for existing usernames and valid email format).
3. Upon successful registration, provide feedback to the user and redirect them back to the login page, where they can use their new credentials to log in.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 12**

**Objective:**

The objective of this lab is to understand the concept of file handling in Java by implementing a signup page that stores user information in a text file. We aim to learn how to write records to a file and retrieve them, emphasizing the use of classes like FileWriter, BufferedWriter, FileReader, and BufferedReader.

**Required Equipment / tools:**

* Eclipse
* JDK (Java Development Kit)

**Introduction:**

### **File Handling in Java**

In Java, file handling is accomplished using the File class, which provides methods for creating, deleting, and renaming files and directories. The File class represents file and directory pathnames, which can be absolute or relative.

#### **File Input and Output:**

* **Reading Files**:
  + FileReader: Used for reading text files in the system's default encoding.
  + FileInputStream: Used for reading binary files and text files with special characters.
* **Writing Files**:
  + FileWriter: Used for writing text files.
  + BufferedWriter: Used for buffering the output to write efficiently.

**Procedure:**

This section includes code for a simple user signup system that records user information into a text file.

package managmentSystem;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class Connection {

String filename = "data.txt";

public void WriteToFile(String record) throws IOException {

FileWriter file = new FileWriter(filename, true);

BufferedWriter writer = new BufferedWriter(file);

writer.write(record);

writer.newLine();

writer.close();

}

public String[][] readFile() throws IOException {

FileReader file = new FileReader(filename);

BufferedReader buffer = new BufferedReader(file);

String line = null;

int i = 0;

String[] records = new String[6];

String data[][] = new String[2][];

while ((line = buffer.readLine()) != null) {

records = line.split(",");

data[i] = records;

i++;

}

return data;

}

}

### **Code for Signup Page:**

This section contains the JavaFX implementation for the signup interface, allowing users to enter their information and store it in a file.

package managmentSystem;

import java.io.IOException;

import java.time.LocalDate;

import javafx.application.Application;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.DatePicker;

import javafx.scene.control.Label;

import javafx.scene.control.ListView;

import javafx.scene.control.RadioButton;

import javafx.scene.control.TextField;

import javafx.scene.control.ToggleGroup;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

public class Signup extends Application {

Stage window;

Scene scene;

Connection con = new Connection();

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Signup Screen");

window.setHeight(600);

window.setWidth(400);

window.setResizable(false);

addComponents();

window.setScene(scene);

window.show();

}

private void addComponents() {

Label name = new Label("Name");

TextField ntext = new TextField();

Label email = new Label("Email");

TextField etext = new TextField();

Label gender = new Label("Gender");

ToggleGroup group = new ToggleGroup();

RadioButton rmale = new RadioButton("Male");

RadioButton rfemale = new RadioButton("Female");

rmale.setToggleGroup(group);

rfemale.setToggleGroup(group);

Label edu = new Label("Education");

ObservableList<String> items = FXCollections.observableArrayList(

"Phd", "Master", "Graduate", "Intermediate", "Matric");

ListView<String> eduList = new ListView<>(items);

Label loc = new Label("Location");

ComboBox<String> locList = new ComboBox<>();

locList.getItems().addAll("Karachi", "Islamabad", "Multan", "Lahore", "Peshawar");

Label dob = new Label("DOB");

DatePicker date = new DatePicker();

date.setValue(LocalDate.now());

Button btnsignup = new Button("Sign up");

Button btnclear = new Button("Clear");

GridPane layout = new GridPane();

layout.setPadding(new Insets(20));

layout.setVgap(10);

layout.add(name, 0, 1);

layout.add(ntext, 1, 1);

layout.add(email, 0, 2);

layout.add(etext, 1, 2);

layout.add(gender, 0, 3);

layout.add(rmale, 1, 3);

layout.add(rfemale, 1, 3);

layout.setMargin(rfemale, new Insets(0, 0, 0, 80));

layout.add(edu, 0, 4);

layout.add(eduList, 1, 4);

layout.add(loc, 0, 5);

layout.add(locList, 1, 5);

layout.add(dob, 0, 6);

layout.add(date, 1, 6);

layout.add(btnsignup, 1, 7);

layout.add(btnclear, 1, 7);

layout.setMargin(btnclear, new Insets(0, 0, 0, 80));

btnsignup.setOnAction(new EventHandler<ActionEvent>() {

@Override

public void handle(ActionEvent event) {

String record = "";

String rgender = "";

record += ntext.getText() + ", ";

record += etext.getText() + ", ";

if (rmale.isSelected())

rgender = "male";

else

rgender = "female";

record += rgender + ", ";

record += eduList.getSelectionModel().getSelectedItem() + ", ";

record += locList.getSelectionModel().getSelectedItem() + ", ";

record += date.getValue() + ", ";

try {

con.WriteToFile(record);

System.out.println("done");

} catch (IOException e) {

System.out.println("error");

e.printStackTrace();

}

}

});

scene = new Scene(layout);

}

}

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

### **Scenario:**

You are developing a user registration system for an online platform that collects personal details from users. The system needs to store this information securely for future access and verification.

### **Task Description:**

Write a Java program that:

* Implements a GUI for user registration that collects the following details: Name, Email, Gender, Education, Location, and Date of Birth.
* Uses file handling to write the collected user data into a text file called users.txt.
* Ensures that each user's details are stored in a structured format, with fields separated by commas.
* Provides functionality to read back and display the stored user records from the users.txt file when requested.

|  |
| --- |
|  |

**Task #2**

### **Scenario:**

You are creating an application to track attendance for a community event. The application should allow users to register their attendance and store their details in a file for later review.

### **Task Description:**

Write a Java program that:

* Develops a user interface to collect the following information from attendees: Name, Email, and Attendance Status (Present/Absent).
* Stores the data in a text file named attendance.txt with each record on a new line.
* Implements a feature to read and display the attendance records from attendance.txt, showing the total number of attendees and their respective attendance status.

|  |
| --- |
|  |

**Task #3**

### **Scenario:**

You are tasked with building a simple product inventory management system for a small retail store. The system should keep track of products and their stock levels.

### **Task Description:**

Write a Java program that:

* Creates a GUI to input product details such as Product Name, Quantity, Price, and Description.
* Saves the product information into a text file called inventory.txt, ensuring each field is separated by commas.
* Includes functionality to read and display all products stored in inventory.txt, providing an overview of the current inventory and allowing the user to search for a specific product by name.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 13**

**Objective:**

The objective of this lab session is to understand the concept of UML class diagrams, focusing on the relationships between classes, and to grasp the principles of inheritance in Java. We will explore how UML diagrams can visually represent class structures and how inheritance enhances code reuse and specialization in object-oriented programming.

**Required Equipment / tools:**

* Microsoft Visio
* Eclipse
* IntelliJ
* NetBeans
* JDK (Java Development Kit)

**Introduction:**

**Unified Modeling Language (UML):**

The Unified Modeling Language (UML) is a family of graphical notations supported by a unified meta-model that aids in describing and designing software systems, particularly those built using the object-oriented (OO) style. As a relatively open standard, UML is managed by the Object Management Group (OMG), an open consortium of companies. It emerged from the unification of various object-oriented graphical modeling languages that gained prominence in the late 1980s and early 1990s.

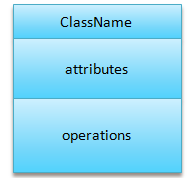
### **Key Concepts:**

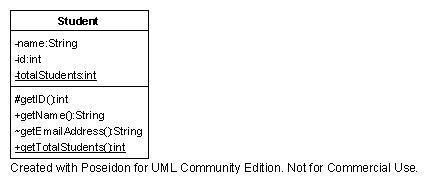
1. **Class Diagram**: A visual representation of the classes in a system and their relationships.
2. **Superclass**: The class from which properties and methods are inherited.
3. **Subclass**: The class that inherits attributes and behaviors from the superclass.
4. **Multilevel Inheritance**: A subclass can serve as a superclass for another subclass, creating a hierarchy of classes.
5. **Method Overriding**: A subclass can provide a specific implementation of a method that is already defined in its superclass.

### **Class Diagram:**

A class diagram describes the types of objects in a system and the various static relationships that exist among them. It illustrates the properties and operations of each class, as well as the constraints that apply to how objects are connected. Class diagrams are widely utilized and encompass a broad range of modeling concepts.

Each class in a class diagram is represented as a rectangle divided into three sections: the top section contains the class name, the middle section lists its attributes, and the bottom section outlines its methods. This structured format effectively conveys essential information about each class in a clear and digestible manner.





**Figure 15.3: Class Example**

**Figure 15.2: class**

**Representing a Class:**

In a class diagram, classes are depicted with their attributes and methods, providing a comprehensive overview of their roles and functionalities within the system. This representation allows for easy identification of each class's characteristics and operations.

Graphically, a **class** is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Use italics for an abstract class name and underline for static members.

**Attributes** are usually listed in the form:

*attributeName : Type*

**Operations** describe the class behavior and appear in the third compartment.

*visibility name (parameters) : return\_type*

**Visibility** can be represented as follows

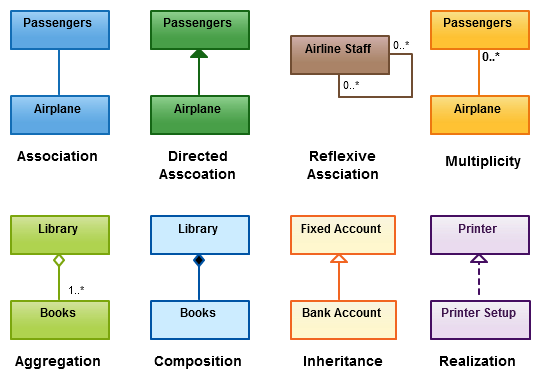
+ public

# protected

- private

/ derived

### **Relationships in Class Diagrams:**



Class diagrams also illustrate the relationships between classes, including:

* **Inheritance** (is-a relationships)
* **Associations** (uses-a relationships)
* **Aggregations or Compositions** (whole-part relationships)

These relationships are crucial for defining how classes interact with one another within the system.

### **Microsoft Visio:**

Microsoft Visio is a powerful diagramming and vector graphics application that is part of the Microsoft Office suite. Initially introduced in 1992 by Shapeware Corporation and acquired by Microsoft in 2000, Visio provides a user-friendly interface for creating various types of diagrams, including class diagrams. This makes it an invaluable tool for software design and documentation.

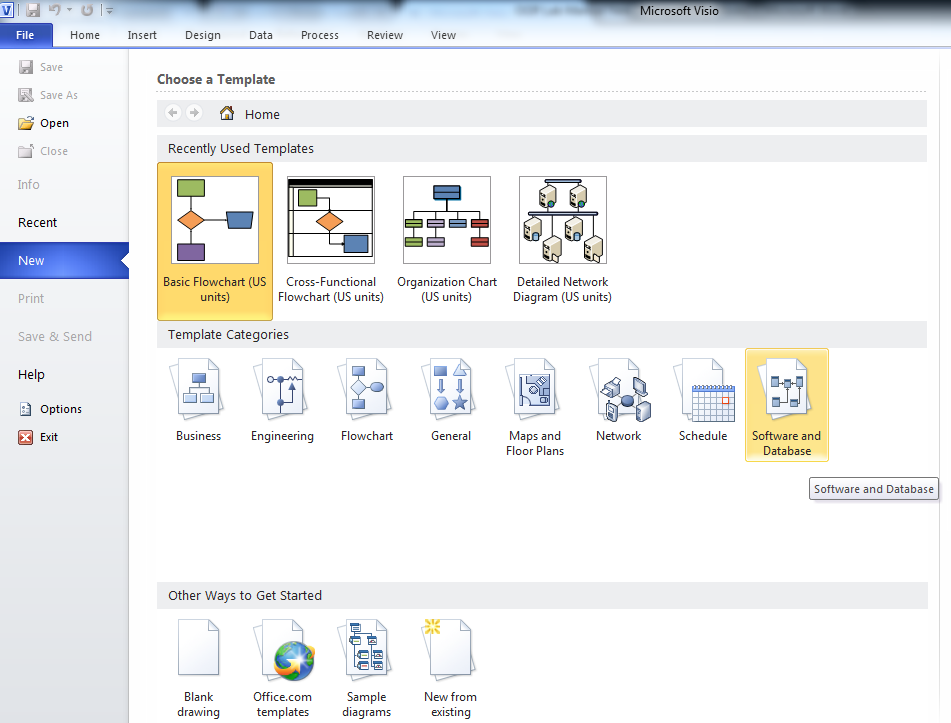
**Procedure:**

We will use Microsoft Visio in this lab to create class diagram.

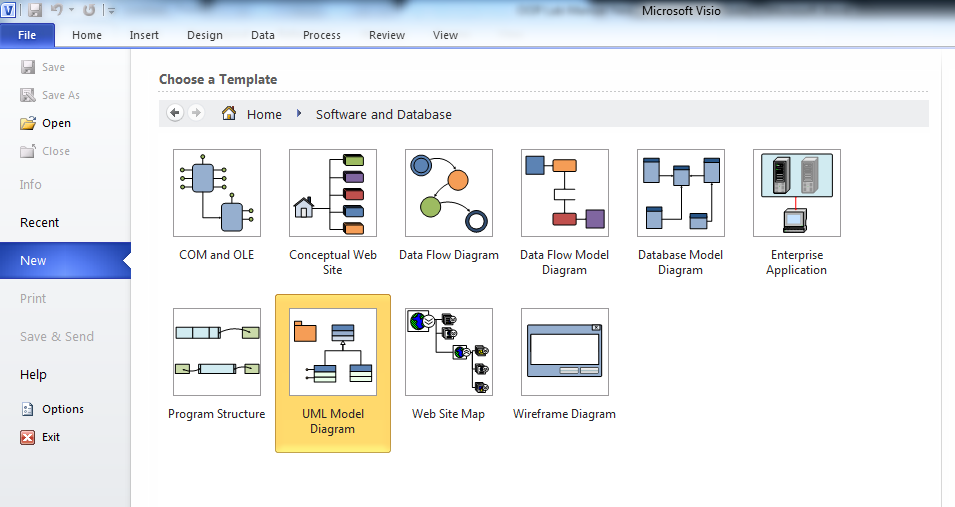
**Creating Class diagram on Visio**

Install the Microsoft Visio on the system. The setup for the software is available on software library.

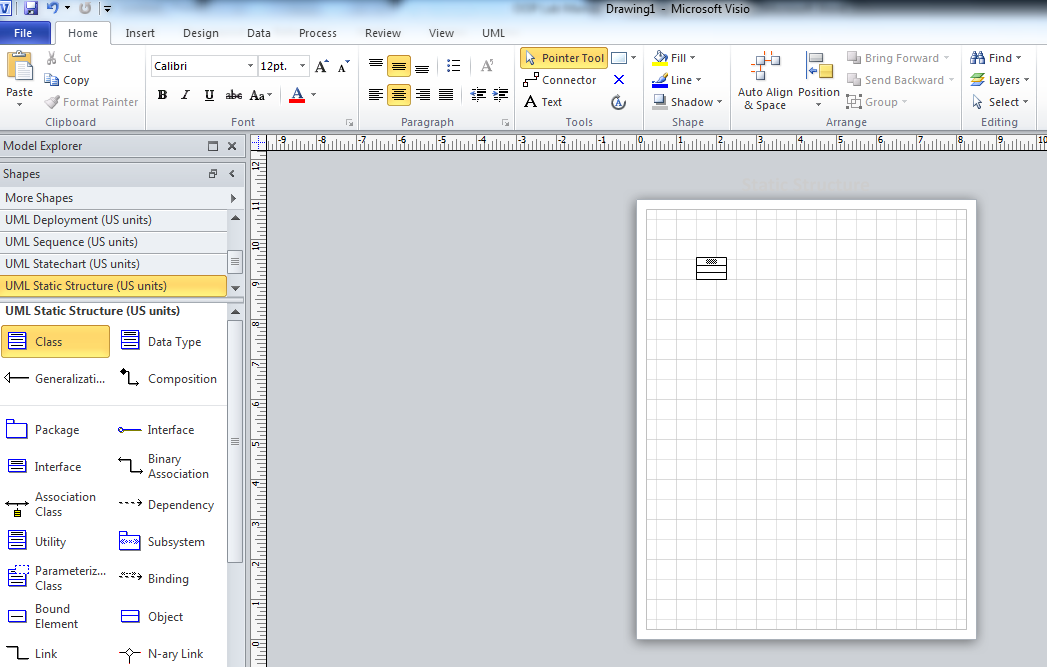
After installation, open Visio. Click on New and then choose option “Software and Database” from the start page.



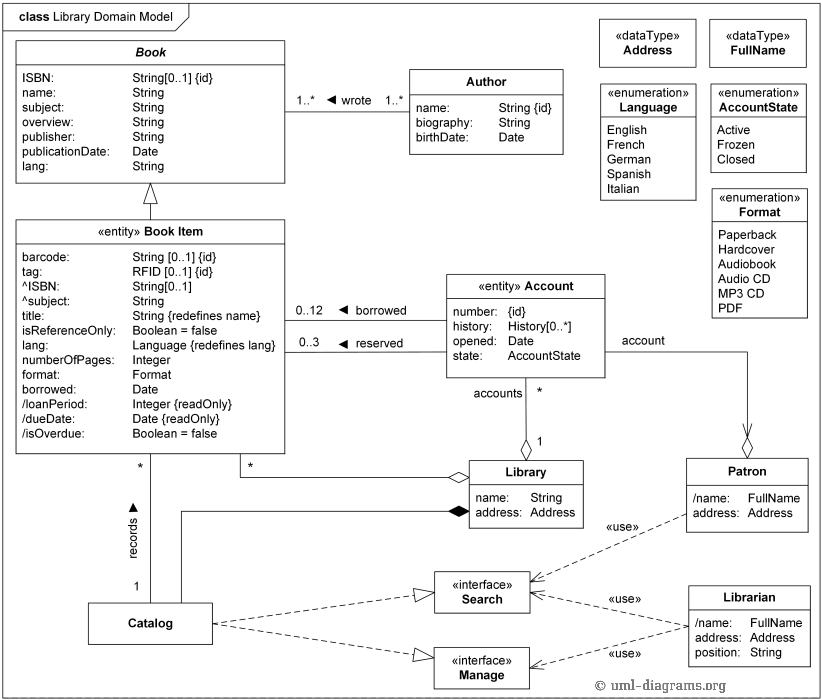
In the next page, select UML Model Diagram.



All the Class diagram elements are available on the left side column. Drag the element and drop it on the sheet. Use connectors as required.



Draw the class diagram given below on Visio.

****

**Expected Outcome:**

|  |
| --- |
|  |

**Task #1**

### Scenario 1: Hospital Management System Class Diagram

**Task Description**:  
Using Microsoft Visio, draw a class diagram for the Hospital Management System based on the following details provided by the client.

* **Entities**:
  + **Person**: Represents individuals associated with the hospital, with derived attributes for **name** (composed of title, first name, middle name, last name) and **address**.
  + **Patient**: Inherits from Person, with unique attributes such as **patient number**, **date of admission**, **blood group**, **sickness**, **allergies**, and **prescriptions**. A derived attribute for **age** is calculated from the birth date.
  + **Staff**: Also inherits from Person, categorized into **Operations Staff** (Doctors and Nurses) and **Administrative Staff**.
  + **Doctor**: A subclass of Operations Staff, with attributes for **specialty**, **designation**, and **room location**. Doctors can **examine**, **provide prescriptions**, and **perform surgeries**.
  + **Nurse**: Another subclass of Operations Staff that works alongside Doctors in patient treatment.
  + **Ward**: Represents divisions within the hospital, differentiated by gender (male and female wards), with attributes for **unique name** and **fixed capacity**.
* **Relationships**:
  + Illustrate the associations between Patients, Doctors, Nurses, and Wards, specifying that a Patient can be treated by 1 or 2 Doctors, and a Ward can have multiple Patients.

|  |
| --- |
|  |

**Task #2**

### **Scenario:** Medical Records Management

**Task Description**:  
Develop a class diagram in Microsoft Visio for managing medical records in the Hospital Management System.

* **Entities**:
  + **MedicalRecord**: Represents the records for each Patient, containing attributes such as **record ID**, **date of record**, **details of treatment**, and **medications prescribed**.
  + **Treatment**: Linked to MedicalRecord, with attributes for **treatment ID**, **type of treatment**, and **date of treatment**.
  + **Medication**: Linked to MedicalRecord, containing attributes for **medication ID**, **name**, **dosage**, and **frequency**.
* **Relationships**:
  + Show that each Patient can have multiple MedicalRecords, and each MedicalRecord can be associated with multiple Treatments and Medications.

|  |
| --- |
|  |

**Task #3**

### **Scenario:** Appointment Scheduling

**Task Description**:  
Create a class diagram using Microsoft Visio for the appointment scheduling component of the Hospital Management System.

* **Entities**:
  + **Appointment**: Represents a scheduled time for a Patient to see a Doctor, with attributes for **appointment ID**, **date**, **time**, and **status** (confirmed, canceled).
  + **Patient**: Inherits from the Person class and has a relationship with Appointment.
  + **Doctor**: Has attributes such as **doctor ID** and **availability**, and is linked to appointments.
* **Relationships**:
  + Illustrate that a Patient can have multiple Appointments, and a Doctor can have multiple Appointments scheduled with different Patients. Include the necessary associations to show how availability affects scheduling.

|  |
| --- |
|  |

**Discussion and analysis of results:**

**Conclusion:**

**Lab Session 14 (Open-Ended Lab)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assessment of Open Ended Lab** / CCA / PBL | | | | |
| **Blooms Taxonomy** | **GAs** | **Knowledge Profile** | **EAs** | WPs |
| P1  P2  P3  P4  P5 | GA-1  GA-2  GA-3  GA-4  GA-5  GA-6  GA-7  GA-18  GA-9  GA-10 | WK1  WK2  WK3  WK4  WK5  WK6  WK7  WK8 | EA1  EA2  EA3  EA4  EA5 | WP1  WP2  WP3  WP4  WP5  WP6  WP7 |

**Title: Dynamic Data Structures: Building a Real-Time Social Media Feed**

**Motivation:**

In the age of social media and instantaneous communication, the efficient management and presentation of dynamic data are of utmost importance. The primary objective of this laboratory is to offer students a practical opportunity to engage in the creation, execution, and enhancement of data structures capable of managing constant data flows, like those found in social media platforms. The laboratory fosters innovation, critical thinking, and the hands-on utilization of theoretical concepts in the realm of data structures.

**Concept (Problem Statement):**

Students will design and implement a data structure to simulate a real-time social media feed. The feed must handle operations like adding new posts, deleting old posts, updating posts, and efficiently retrieving the latest posts. The lab will also involve implementing features such as sorting posts by different criteria (e.g., timestamp, likes, relevance) and ensuring the system performs efficiently under various conditions.

**Design:** Students are required to follow the design constraints set out for the implementation of OEL:

* Introduction and Requirements
* Data Structure Selection
* Basic Implementation
* Performance Testing and Analysis
* Optimization and Advanced Features
* Extensions and Creativity

**Open Ended Lab Assessment Rubrics**

Course Code and Title: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria and Scales** | | | |
| Excellent  (10-8) | Good  (7-5) | Average  (4-2) | Poor  (1-0) |
| **Criterion 1:** Understanding the Problem: How well the problem statement is understood by the student | | | |
| Understands the problem clearly and clearly identifies the underlying issues. | Adequately understands the problem and identifies the underlying issues. | Inadequately defines the problem and identifies the underlying issues. | Fails to define the problem adequately and does not identify the underlying issues. |
| **Criterion 2:** Research: The amount of research that is used in solving the problem | | | |
| Contains all the information needed for solving the problem | Good research, leading to a successful solution | Mediocre research which may or may not lead to an adequate solution | No apparent research |
| **Criterion 3:**  **Class Diagram:** The completeness of the class diagram | | | |
| Class diagram with complete notations | Class diagram with incomplete notations | Class diagram with improper naming convention and notations | No Class diagram |
| **Criterion 4:**  **Code:** How complete and accurate the code is along with the assumptions | | | |
| Complete Code according to the class diagram of the given case with clear assumptions | Incomplete Code according to the class diagram of the given case with clear assumptions | Incomplete Code according to the class diagram of the given case with unclear assumptions | Wrong code and naming conventions |
| **Criterion 5:**  **Report**: How thorough and well organized is the solution | | | |
| All the necessary information clearly organized for easy use in solving the problem | Good information organized well that could lead to a good solution | Mediocre information which may or may not lead to a solution | No report provided |

Total marks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Teacher’s signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Open Ended Lab Assessment Rubrics**

Course Code and Title: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria and Scales | | | | |
| Excellent  (10-8) | Good  (7-5) | Average  (4-2) | Poor  (1-0) | Total Marks  10 |
| Criterion 1: Understanding the Problem: How well the problem statement is understood by the student | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| Criterion 2: Research: The amount of research that is used in solving the problem | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| Criterion 3: Class Diagram: The completeness of the class diagram | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| Criterion 4: Code: How complete and accurate the code is along with the assumptions | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| Criterion 5: Report: How thorough and well organized is the solution | | | | |
| (10-8) | (7-5)% | (4-2)% | (1-0)% |  |
| Total | | | | (\_\_\_\_/5) |

Total marks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Teacher’s signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_