IQRA UNIVERSITY IU

Department of Computer Science

Lab Manual

CMC112-L Object Oriented Programming

Instructor's Name: ______________________________

Student's Name: ________________________________

Roll No.: _______________ Batch: _________________

Semester: _______________ Year: __________________

Department: ____________________________________

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

Department of Computer Science

Lab Manual

CMC112-L Object Oriented Programming

Prepared By:

OOP Lab Team

Reviewed / Approved By:

____

Faculty of Engineering Sciences & Technology

Iqra University

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

Table of Contents

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

CONTENTS

11.		Understanding and implementing various layouts and charts in JavaFX to create enhanced and interactive user interfaces.		
12.		Understanding JavaFX applications: Design and event handling for interactive user interfaces.		
13.		Understanding data storage and retrieval in a signup page using file handling in Java.		
14.		Assessment of Open Ended lab		
Final Examination				

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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 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.

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

<u>Psychomotor Rubrics Assessment Hardware Lab</u>

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)***

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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/simulat e
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.

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

<u>Psychomotor Rubrics Assessment Software based Lab</u>

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)***

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

<u>Affective Domain Rubrics Assessment</u>

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)

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

<u>Open Ended Lab Assessment Rubrics</u>

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

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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: ____________________

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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: ____________________

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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: ______________________

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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)

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

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:

Introduction to Java and Programming Elements:

a) Getting familiar with the Java development kit (JDK). Running your first Java program using CMD and an IDE.
b) To learn Input/output handling on Java console. Understanding variables using primitive and non-primitive data types. Exploring Java's built-in classes.

Required Tools:

- Eclipse
- JDK (Java Development Kit)

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

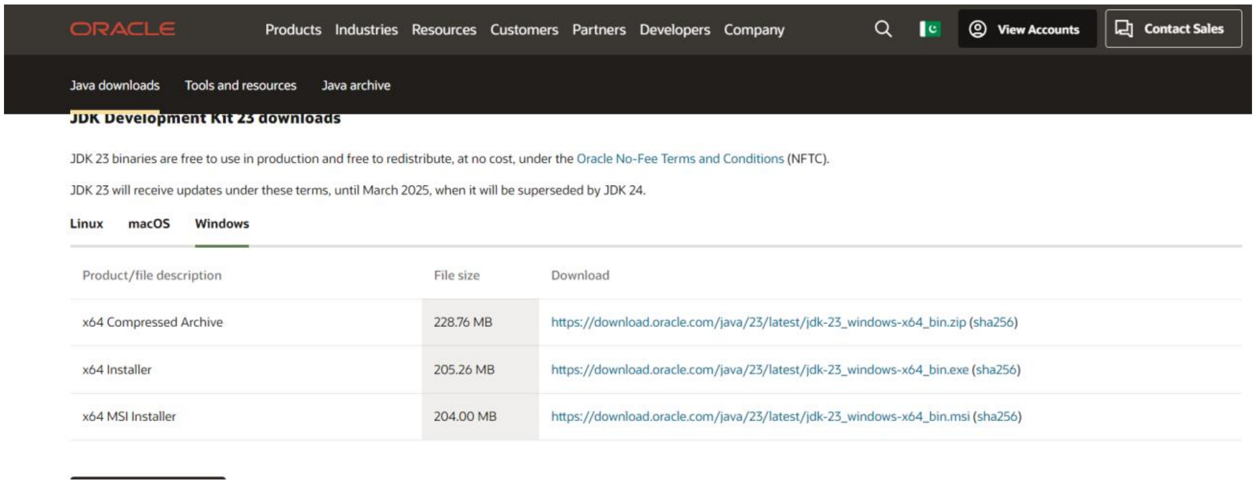
1. Download the JDK from Oracle's website. Choose the right JDK depending upon your system's specifications.

Required IDE:

- JDK (Java Development Kit)
- Eclipse

Procedure:

Step 1: Setup the JDK:



ORACLE
Products Industries Resources Customers Partners Developers Company
View Accounts
Contact Sales
Java downloads Tools and resources Java archive

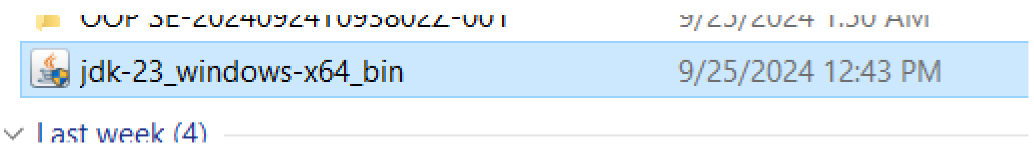
JDK Development Kit 23 downloads

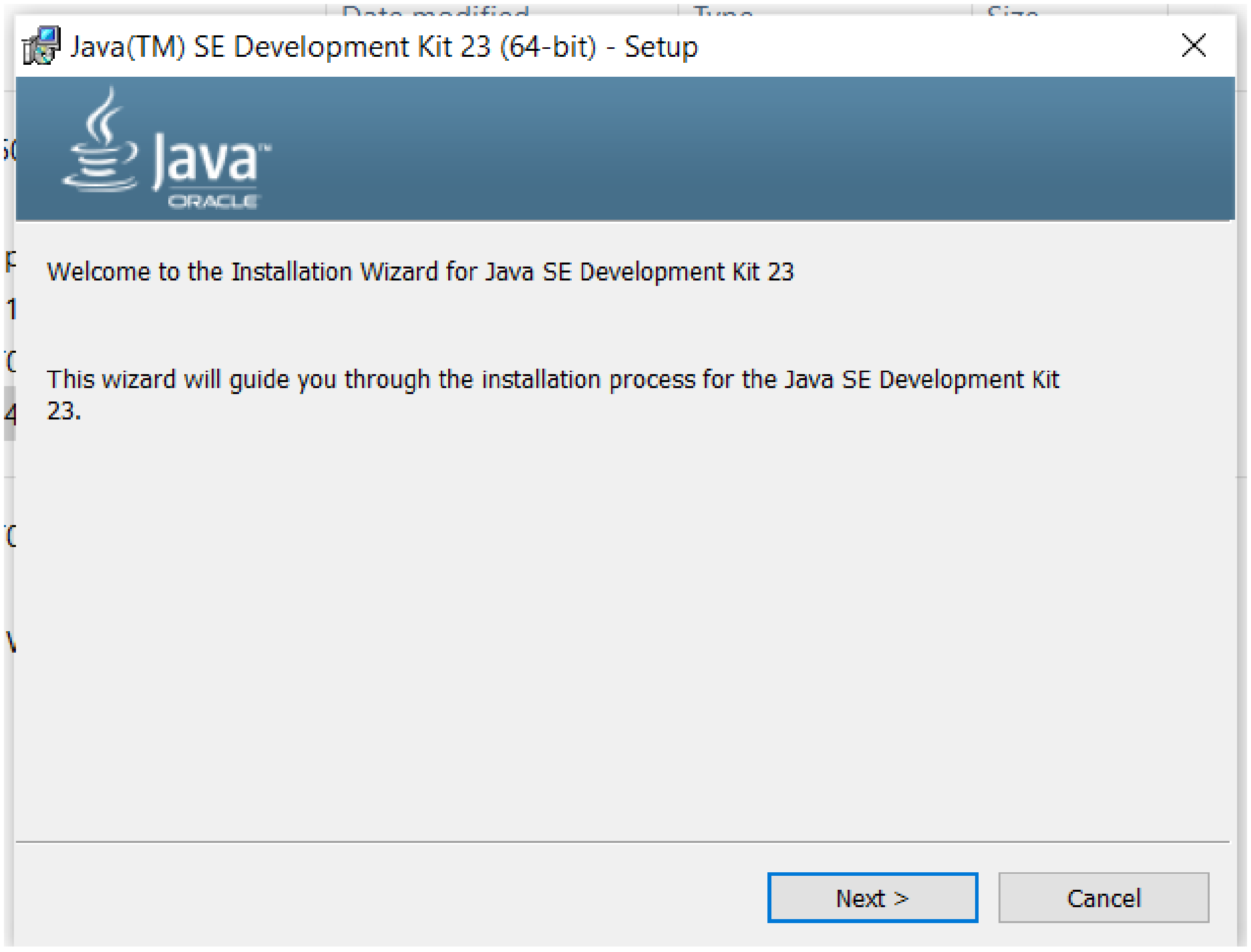
JDK 23 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

JDK 23 will receive updates under these terms, until March 2025, when it will be superseded by JDK 24.

Linux macOS Windows

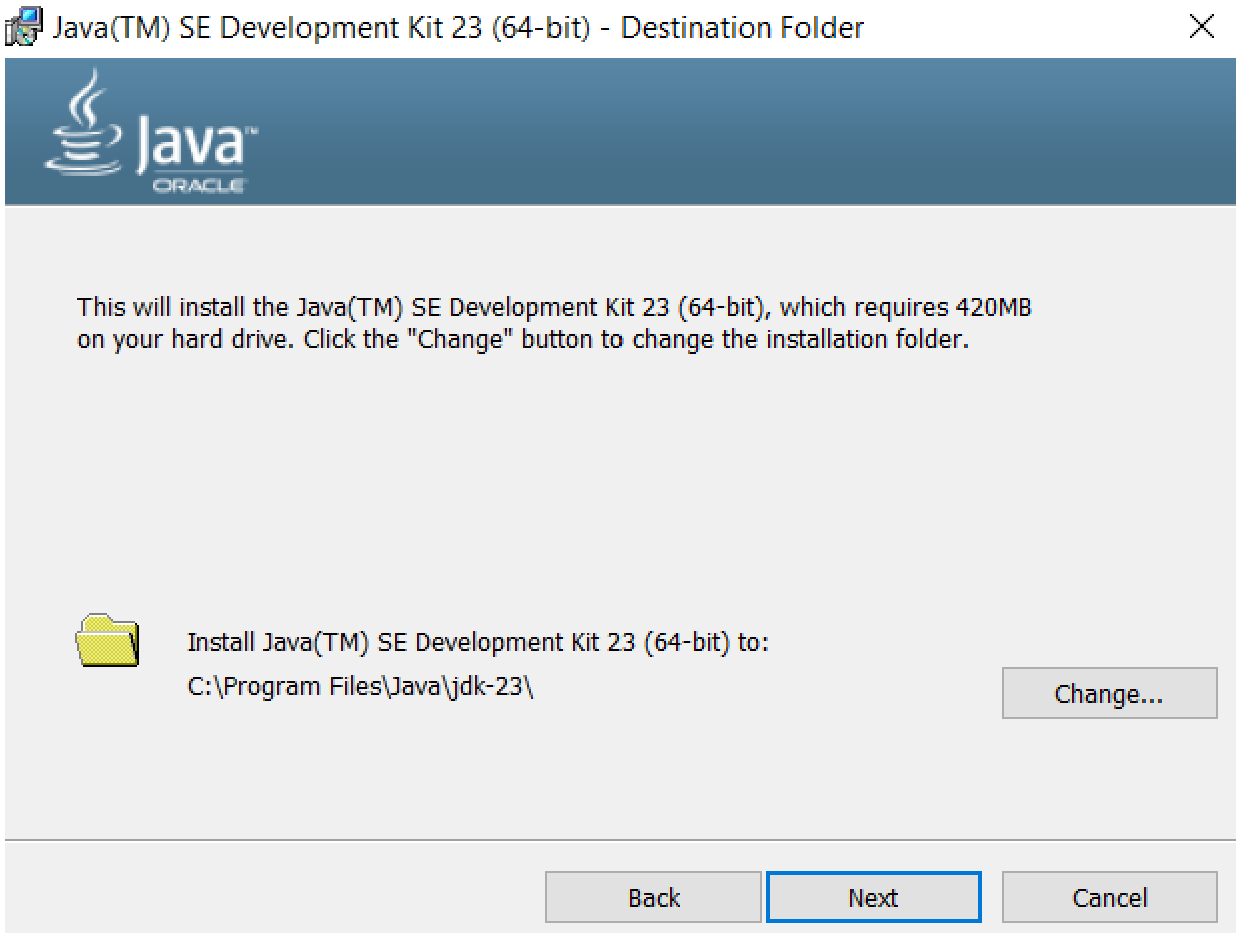
Product/file description	File size	Download
x64 Compressed Archive	228.76 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.zip (sha256)
x64 Installer	205.26 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.exe (sha256)
x64 MSI Installer	204.00 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.msi (sha256)





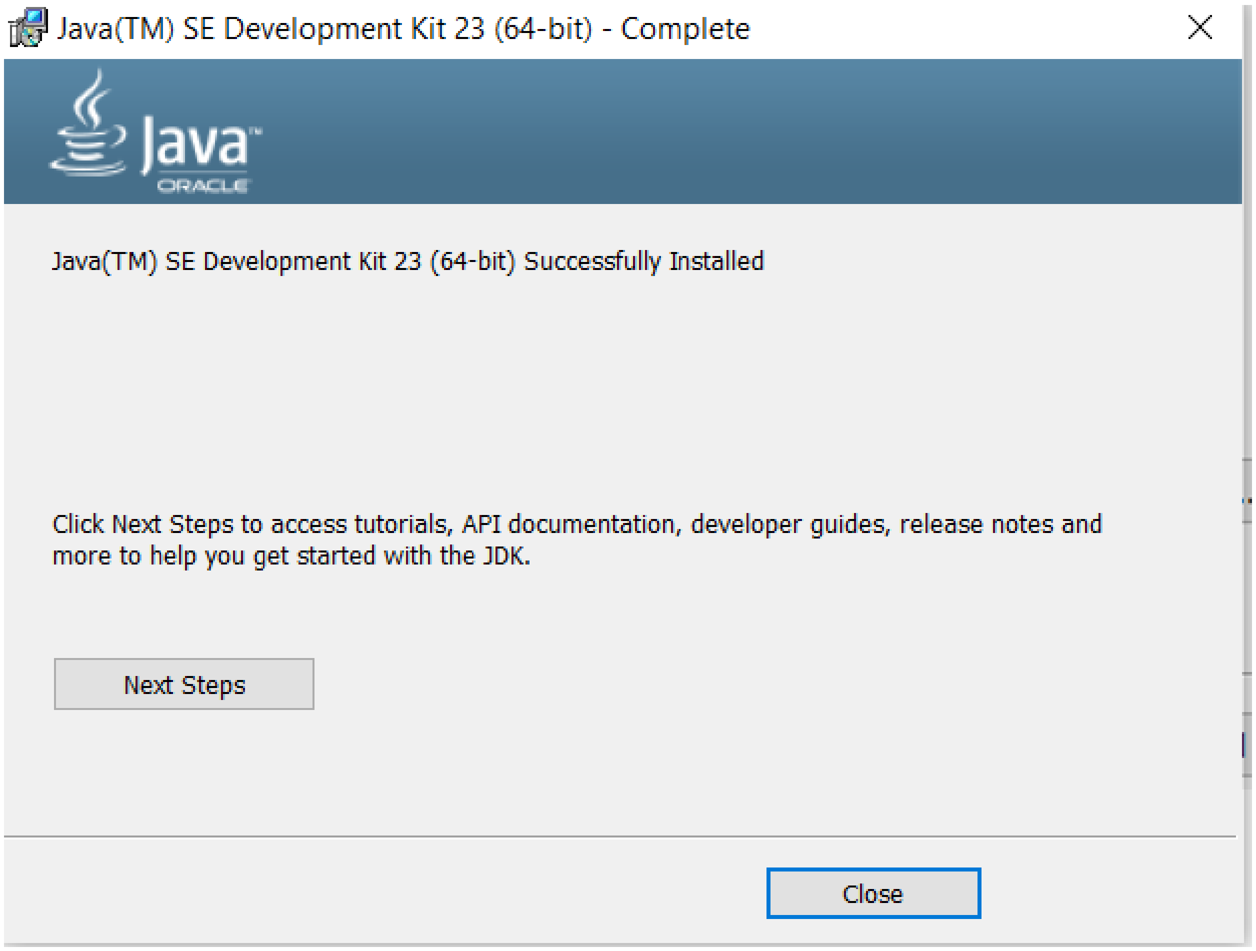
Welcome to the Installation Wizard for Java SE Development Kit 23

This wizard will guide you through the installation process for the Java SE Development Kit 23.


Java(TM) SE Development Kit 23 (64-bit) - Destination Folder

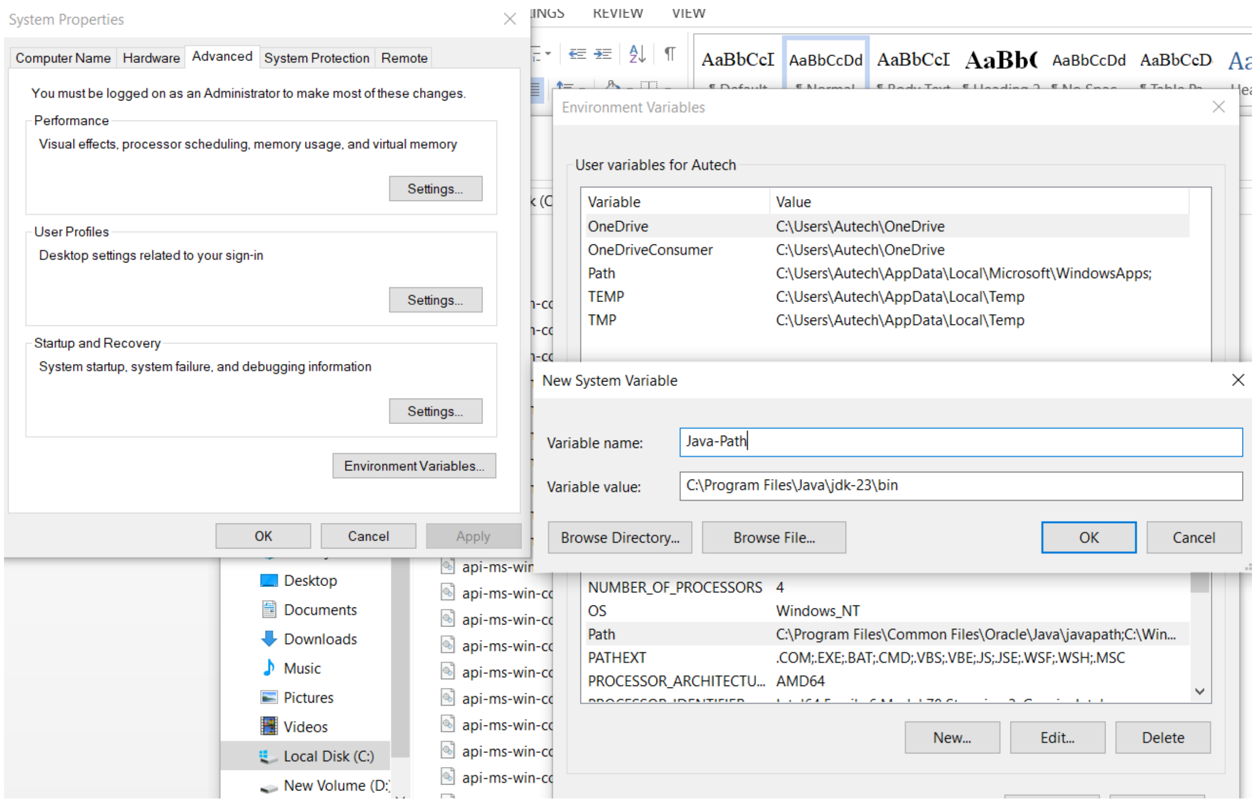
This will install the Java(TM) SE Development Kit 23 (64-bit), which requires 420MB on your hard drive. Click the "Change" button to change the installation folder.

Install Java(TM) SE Development Kit 23 (64-bit) to:
C:\Program Files\Java\jdk-23\


Java(TM) SE Development Kit 23 (64-bit) - Complete

Java(TM) SE Development Kit 23 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.



You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles: Desktop settings related to your sign-in

Startup and Recovery: System startup, system failure, and debugging information

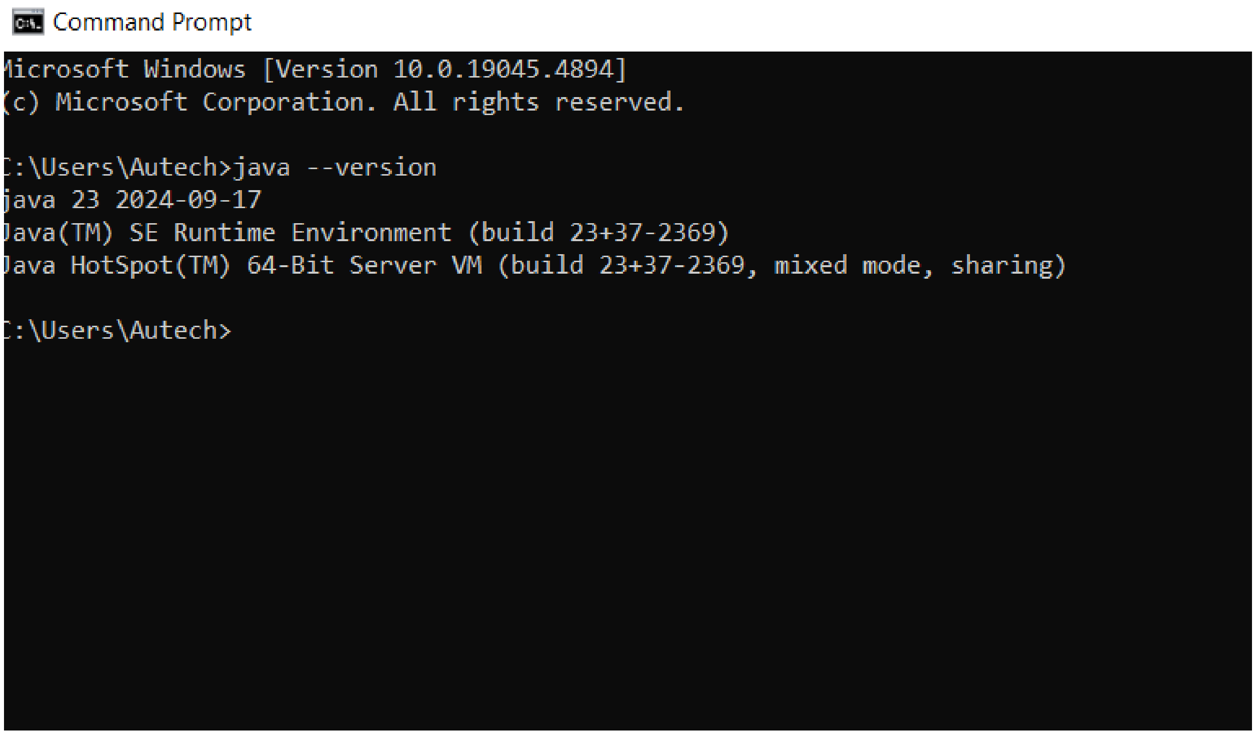
User variables for Autech

Variable	Value
OneDrive	C:\Users\Autech\OneDrive
OneDriveConsumer	C:\Users\Autech\OneDrive
Path	C:\Users\Autech\AppData\Local\Microsoft\WindowsApps;
TEMP	C:\Users\Autech\AppData\Local\Temp
TMP	C:\Users\Autech\AppData\Local\Temp

Variable name: Java-Path

Variable value: C:\Program Files\Java\jdk-23\bin

NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64



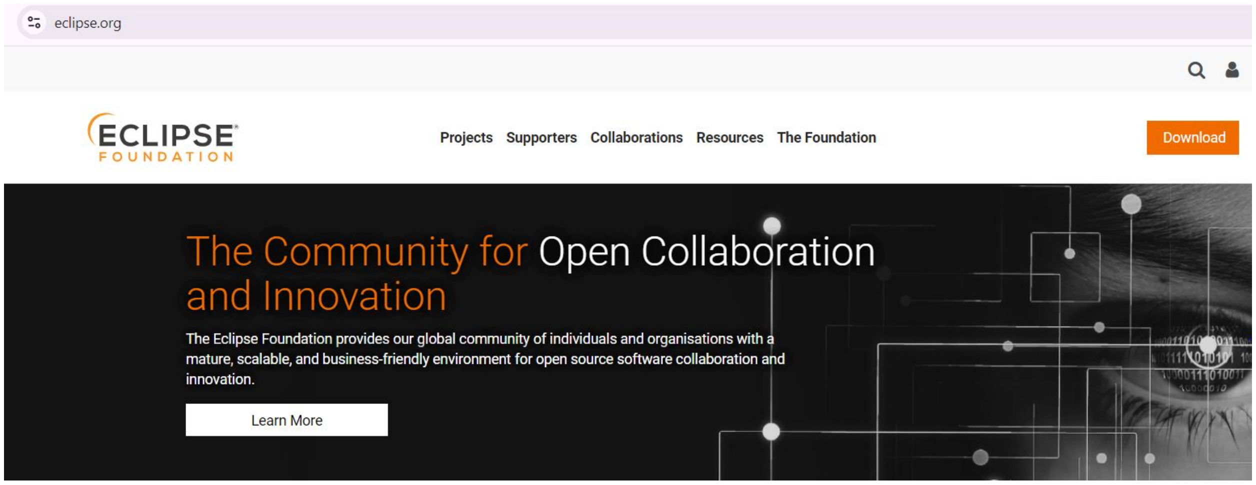
```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Autech>java --version
java 23 2024-09-17
Java(TM) SE Runtime Environment (build 23+37-2369)
Java HotSpot(TM) 64-Bit Server VM (build 23+37-2369, mixed mode, sharing)

C:\Users\Autech>
```

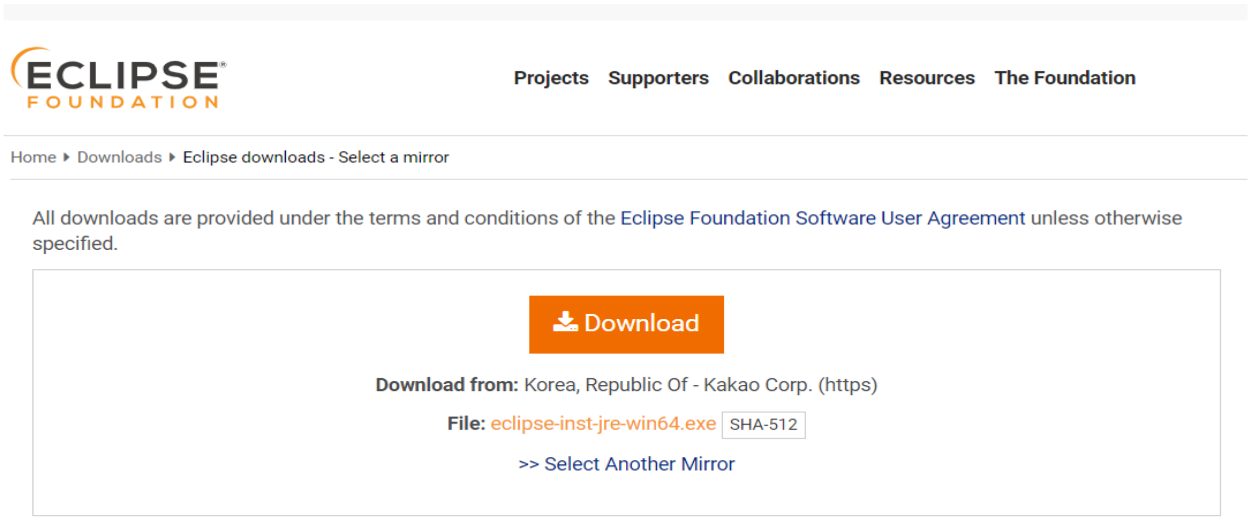
Step 2: Setup the Eclipse IDE

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.



The Community for Open Collaboration and Innovation

The Eclipse Foundation provides our global community of individuals and organisations with a mature, scalable, and business-friendly environment for open source software collaboration and innovation.

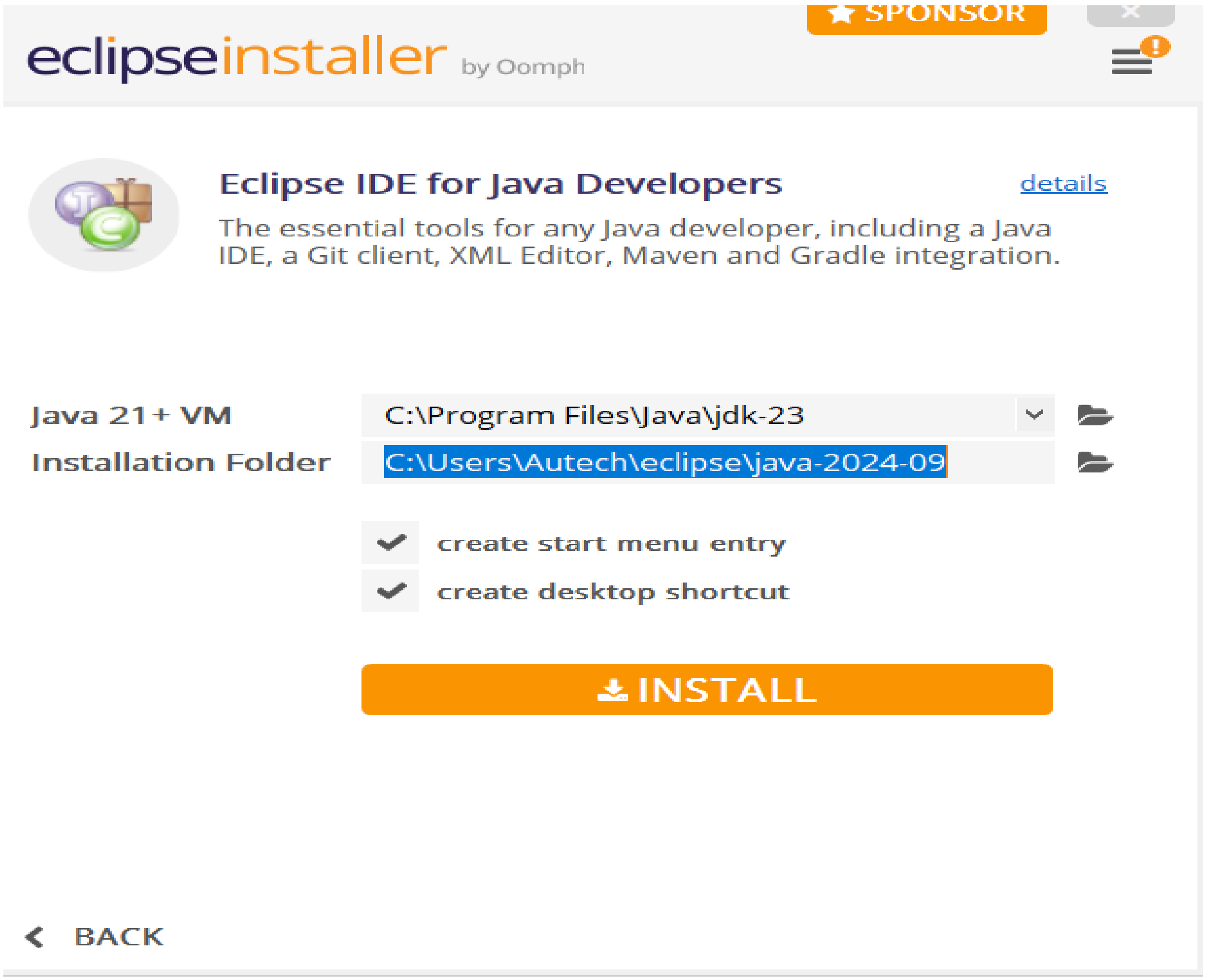


All downloads are provided under the terms and conditions of the Eclipse Foundation Software User Agreement unless otherwise specified.

Download from: Korea, Republic Of - Kakao Corp. (https)

File: eclipse-inst-jre-win64.exe SHA-512

>> Select Another Mirror



Eclipse IDE for Java Developers

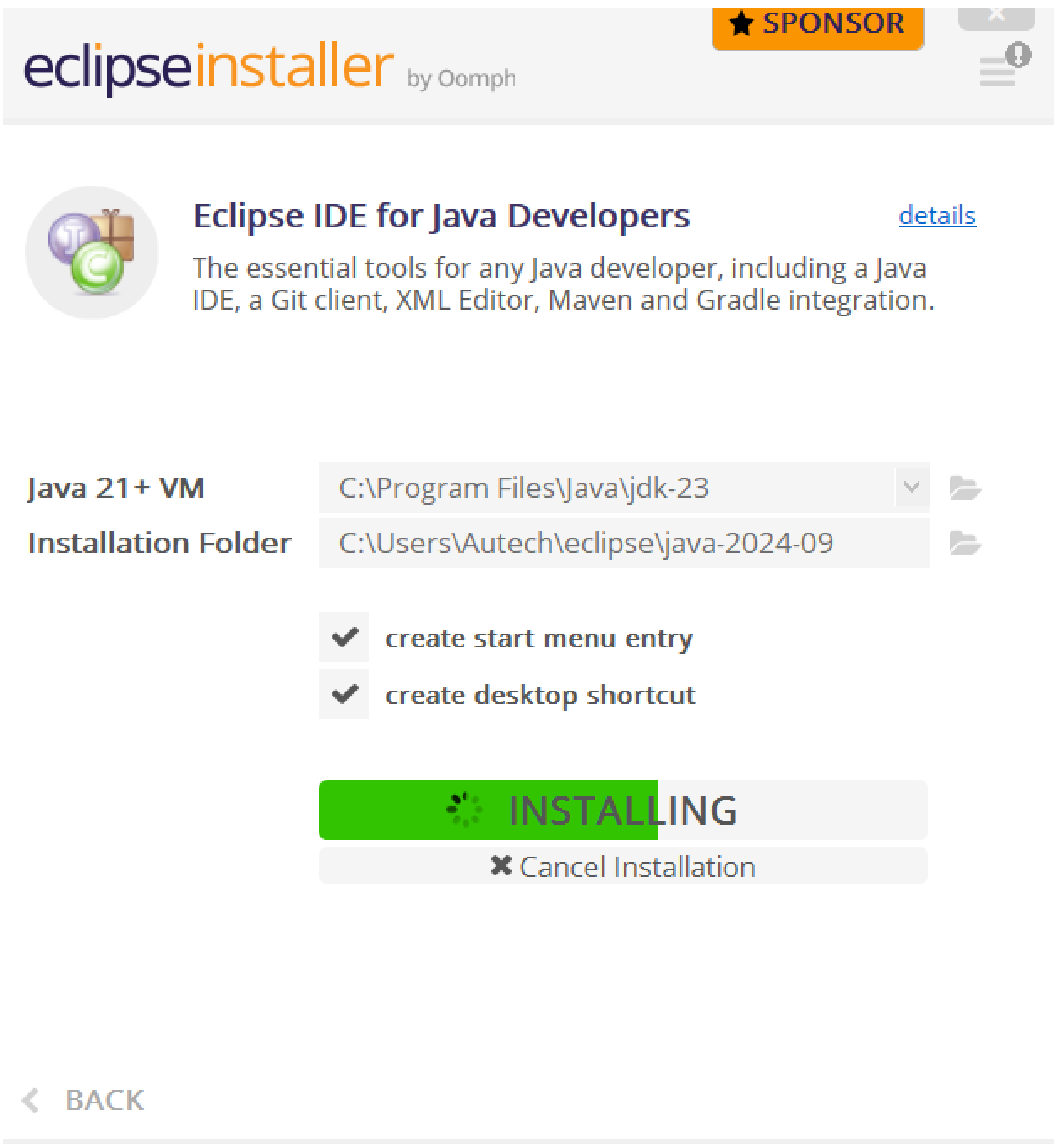
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

Java 21+ VM C:\Program Files\Java\jdk-23

Installation Folder C:\Users\Autech\eclipse\java-2024-09

✓ create start menu entry

✓ create desktop shortcut



Eclipse IDE for Java Developers details

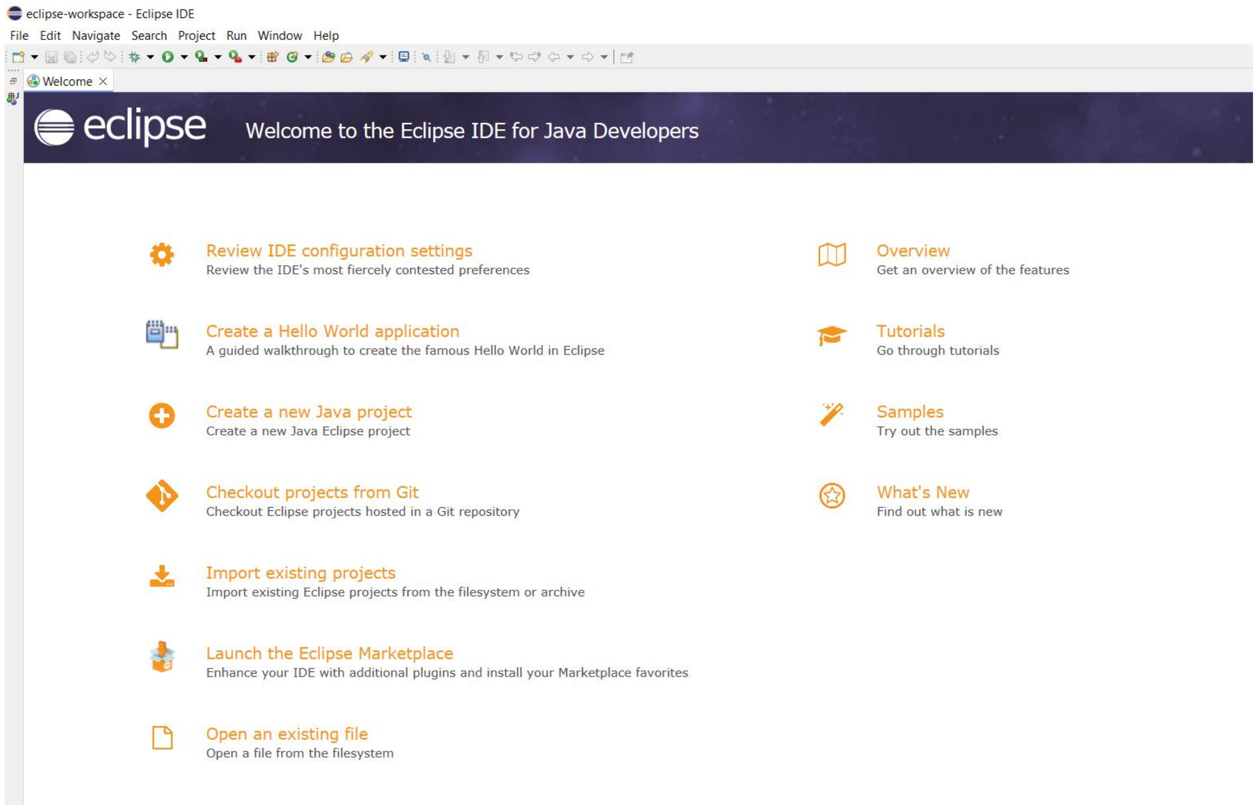
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration.

Java 21+ VM: C:\Program Files\Java\jdk-23

Installation Folder: C:\Users\Autech\eclipse\java-2024-09

✓ create start menu entry

✓ create desktop shortcut


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Welcome
eclipse

Welcome to the Eclipse IDE for Java Developers

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse

Create a new Java project
Create a new Java Eclipse project

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites

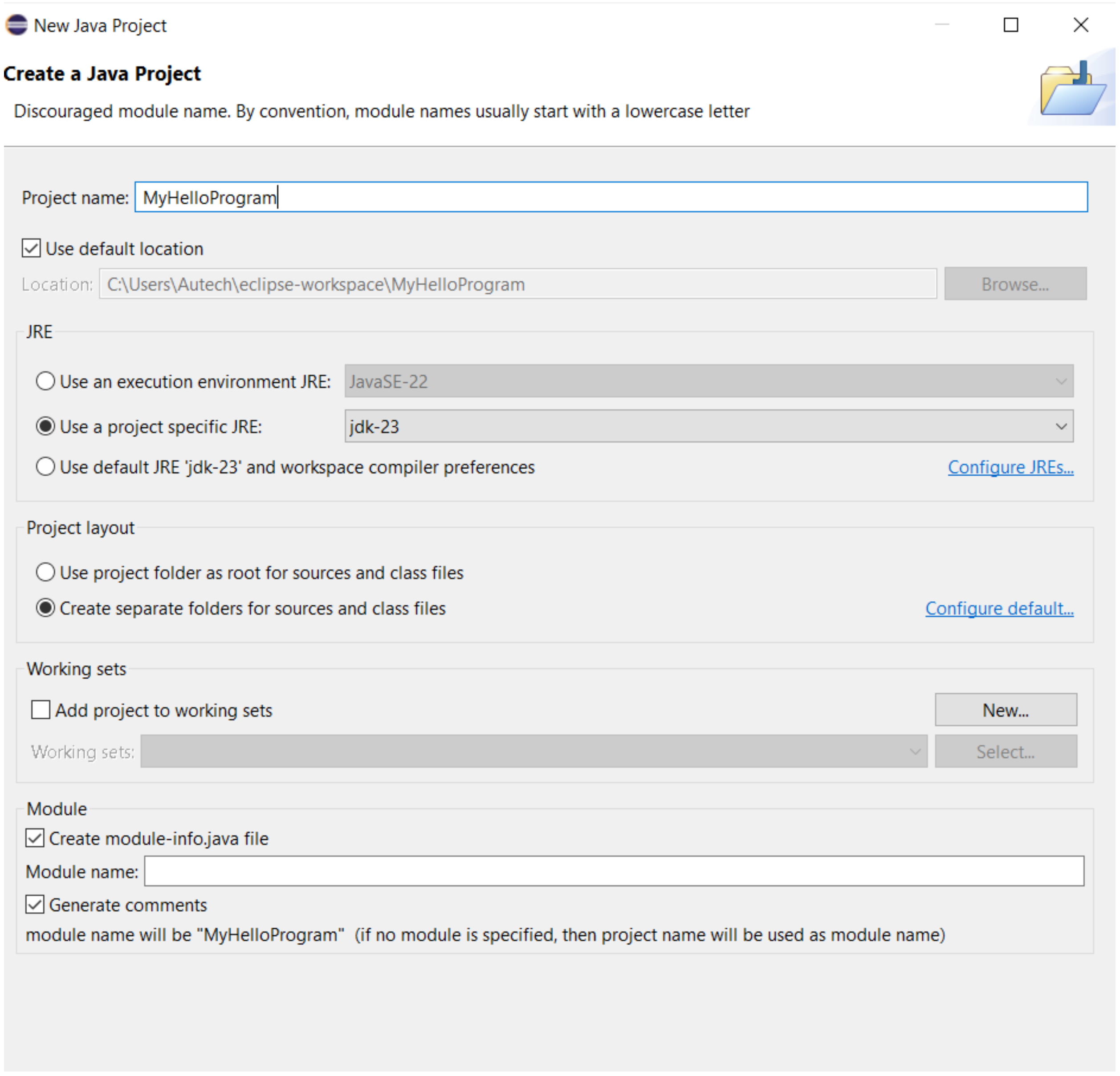
Open an existing file
Open a file from the filesystem

Overview
Get an overview of the features

Tutorials
Go through tutorials

Samples
Try out the samples

What's New
Find out what is new



New Java Project

Create a Java Project

Discouraged module name. By convention, module names usually start with a lowercase letter

Project name: MyHelloProgram

[✓] Use default location

Location: C:\Users\Autech\eclipse-workspace\MyHelloProgram

JRE

○ Use an execution environment JRE: JavaSE-22

● Use a project specific JRE: jdk-23

○ Use default JRE 'jdk-23' and workspace compiler preferences

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

Working sets:

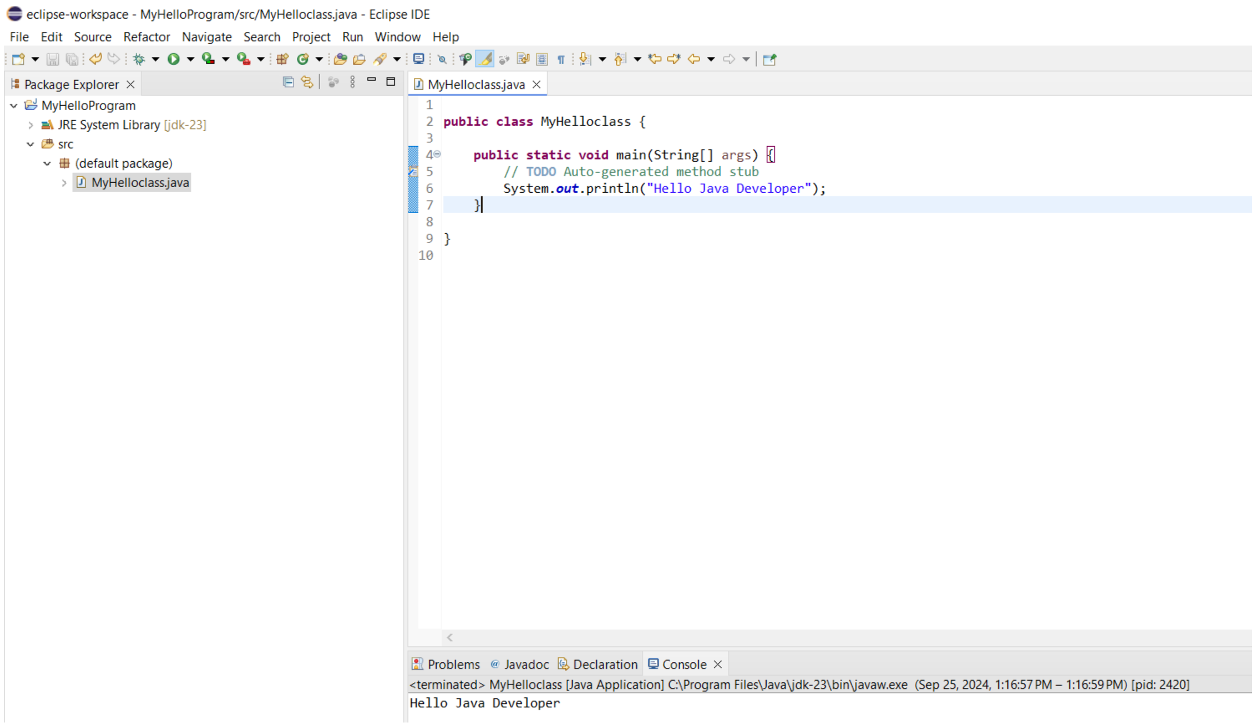
Module

[✓] Create module-info.java file

Module name:

[✓] Generate comments

module name will be "MyHelloProgram" (if no module is specified, then project name will be used as module name)


eclipse-workspace - MyHelloProgram/src/MyHelloclass.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
MyHelloProgram
JRE System Library [jdk-23]
src
(default package)
MyHelloclass.java
Problems Javadoc Declaration Console

```
public class MyHelloclass {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("Hello Java Developer");
	}

}
```

```
<terminated> MyHelloclass [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (Sep 25, 2024, 1:16:57 PM – 1:16:59 PM) [pid: 2420]
Hello Java Developer
```

Understanding Java Console Input and Output

Introduction:

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.

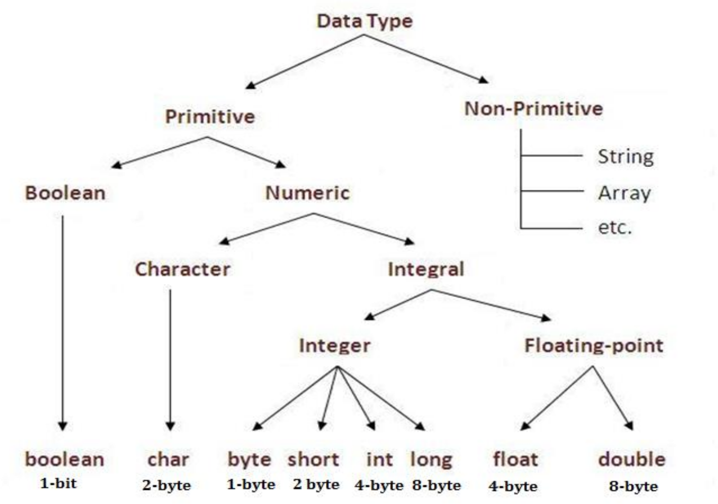


Figure 2.1: Data Types in Java

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;

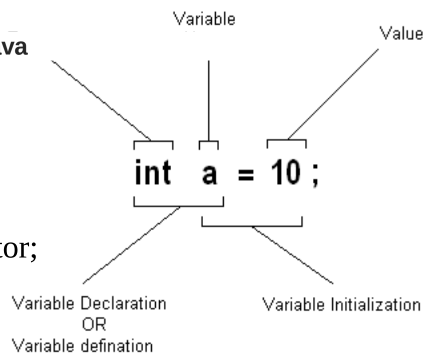


Figure 2.2: Variable Initialization

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 the variable'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.

Procedure:

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

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

        double a = Math.toRadians(45); // Convert degrees to radians
        double 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("\nTrigonometric 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);
    }
}
```

Observation:

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.

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.

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.

1. **Home Task #1**

Interactive Calculator (CMD and IDE)

- **Objective:** Create a simple calculator application that takes input from the user and performs basic arithmetic operations.
- **Instructions:**

1. **CMD Version:** Write a Java program that prompts the user to enter two numbers and an operator (+, -, *, /). Use Scanner to read the input from the console. Perform the requested operation and print the result to the console. Compile and run the program using the command line (CMD).

2. **IDE Version:** Create a new project in Eclipse (or your chosen IDE). Implement the same calculator functionality within the IDE. Run and test the program within the IDE.

2. **Home Task #2**

Data Type Exploration and Type Conversion

- **Objective:** Experiment with different data types and practice type conversion.
- **Instructions:**

1. **Variable Declarations:** Declare variables of different primitive data types (int, float, double, char, boolean). Initialize them with appropriate values.
2. **Output Formatting:** Use System.out.printf() to display the variables in various formats (e.g., with specific decimal places, padding, etc.).
3. **Type Conversion:** Perform explicit type conversions (casting) between different data types. Observe the results and explain any data loss or truncation that occurs. For example:
    - Cast a double to an int.
    - Cast an int to a char.
    - Cast a char to an int.
4. **String Conversion:** Convert numeric data types to strings using String.valueOf() and convert strings to numeric data types using Integer.parseInt(), Double.parseDouble(), etc.
5. **Console Output:** Print the results of all operations to the console with clear explanations.

3. **Home Task #3**

Date and Math Class Usage

- **Objective:** Practice using the Math and Date classes.
- **Instructions:**

1. **Current Date and Time:** Create a Date object to get the current date and time. Print the current date and time to the console.
2. **Math Operations:** Prompt the user to enter a number. Use the Math class to perform the following operations and print the results:
   - Calculate the square root.
   - Calculate the sine, cosine, and tangent.
   - Calculate the power of the number raised to a specified exponent (also get the exponent from the user).
   - Generate a random number between 0 and 1.
3. **Formatted Output:** Use System.out.printf() to format the output of the mathematical operations to a specified number of decimal places.

Discussion and analysis of results:

Conclusion:

Lab Session 2

Objective:

Understanding 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
}
```

2. **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
    }
```

3. **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:

1. **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.");
        }
    }
}
```

2. **Generate a Number Pattern using Nested Loops**

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

```
1
12
123
1234
12345
```

```
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();
        }
    }
}
```

3. **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;
        }
    }
}
```

4. 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.");
        }
    }
}
```

Observation:

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.

1. **Home Task #1**

Enhanced Season Finder with Switch Case

- **Objective:** Modify the provided "Season Finder" program to use a switch statement instead of the if-else-if ladder.
- **Instructions:**

1. **Refactor:** Take the existing SeasonFinder code and replace the if-else-if structure with a switch statement based on the month variable.
2. **Input Validation:** Add input validation to handle invalid month inputs (outside the range 1-12). Print an error message if the input is invalid.
3. **Southern Hemisphere (Bonus):** Add an option for the user to specify whether they are in the Northern or Southern Hemisphere. Adjust the season output accordingly. (For example, if it's January and the user is in the Southern Hemisphere, the season would be Summer).

2. **Home Task #2**

- **Objective:** Create Java programs to generate different number patterns using nested loops.
- **Instructions:**

1. **Inverted Triangle:** Write a program to generate the following pattern:

```
12345
1234
123
12
1
```

2. **Pyramid:** Write a program to generate a pyramid pattern like this (you can adjust the height):

```
    1
   121
  12321
 1234321
123454321
```

1. **Diamond (Bonus):** Combine the pyramid and inverted triangle patterns to create

3. **Home Task #3**

The Number Guessing Game Challenge

- **Objective:** Develop a number guessing game using Java's control flow statements (loops, conditionals), random number generation, and user input. This task focuses on combining different programming concepts to create an interactive application.
- **Instructions:**

1. **Game Setup:**
    - The program should generate a random secret number between 1 and 100 (inclusive). Use java.util.Random to achieve this.
    - The user has a limited number of attempts to guess the number (e.g., 7 attempts).
2. **User Interaction:**
    - Use a Scanner to get the user's guess as input.
    - Provide clear prompts to the user, such as "Enter your guess (1-100):".
3. **Game Logic:**
    - After each guess, provide feedback:
        - "Too high!" if the guess is greater than the secret number.
        - "Too low!" if the guess is less than the secret number.
    - Keep track of the number of attempts.
4. **Winning and Losing:**
    - If the user guesses the correct number, congratulate them and display the number of attempts it took.
    - If the user runs out of attempts without guessing correctly, reveal the secret number and inform them that they've lost.
5. **Bonus Challenges (Optional):**
    - **Difficulty Levels:** Implement different difficulty levels (e.g., easy, medium, hard) that adjust the range of the secret number or the number of allowed attempts.
    - **Play Again:** Add an option for the user to play again without restarting the program.

- **Example Interaction:**

```
Welcome to the Number Guessing Game!
I'm thinking of a number between 1 and 100.
You have 7 attempts to guess it.
Attempt 1: Enter your guess (1-100): 50
Too high!
Attempt 2: Enter your guess (1-100): 25
Too low!
...
Attempt 7: Enter your guess (1-100): 32
Congratulations! You guessed the number in 7 attempts. The number was 32.
Do you want to play again? (y/n):
```

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:

1. 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);
    }
}
```

2. 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;
    }
}
```

3. 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);
    }
}
```

Observation:

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?

1. **Home Task #1**

Enhance the Box Class

- **Objective:** Expand the functionality of the provided Box class.
- **Instructions:**

1. **Add Constructors:** Implement two constructors for the Box class:
   - A default constructor that initializes width, height, and depth to 1.
   - A parameterized constructor that takes three arguments representing width, height, and depth, and initializes the instance variables accordingly.
2. **Add Methods:** Add the following methods to the Box class:
   - getSurfaceArea(): Calculates and returns the surface area of the box.
   - isEqual(Box otherBox): Takes another Box object as input and returns true if the dimensions of the two boxes are equal, and false otherwise.
3. **Test:** In a separate BoxTest class, create several Box objects using both constructors. Test the volume(), getSurfaceArea(), and isEqual() methods. Print the results to the console to verify correctness.

2. **Home Task #2**

Design a Rectangle Class

- **Objective:** Design and implement a Rectangle class from scratch, focusing on geometric calculations.
- **Instructions:**

1. **Class Definition:** Create a class named Rectangle.
2. **Data Members:** Include the following data members (instance variables):
   - length (double): The length of the rectangle.
   - width (double): The width of the rectangle.
3. **Constructors:** Create two constructors:
   - A default constructor that initializes both length and width to 0.
   - A parameterized constructor that takes the length and width as arguments and initializes the instance variables accordingly. Handle invalid input (negative length or width) by setting the dimensions to 0 and printing an error message.
4. **Methods:** Implement the following methods:
   - calculateArea(): Calculates and returns the area of the rectangle.
   - calculatePerimeter(): Calculates and returns the perimeter of the rectangle.
   - isSquare(): Returns true if the rectangle is a square (length equals width), and false otherwise.
   - displayRectangleInfo(): Prints the length, width, area, and perimeter of the rectangle to the console. Also indicate whether the rectangle is a square.
5. **Test:** Create a RectangleTest class to create Rectangle objects using both constructors (including cases with invalid input). Test all the methods you've implemented and print the results to verify correctness. For example, create rectangles with different dimensions, calculate their area and perimeter, and check if they are squares.

3. **Home Task #3**

Bank Account Management

- **Objective:** Create a system for managing bank accounts using classes and objects.
- **Instructions:**

1. **Account Class:** Create a class named Account.
2. **Data Members:** Include the following data members:
   - accountNumber (String)
   - accountHolderName (String)
   - balance (double)
3. **Constructors:** Create a parameterized constructor that takes the account number and account holder's name as input and initializes the balance to 0.
4. **Methods:** Implement the following methods:
   - deposit(double amount): Deposits the specified amount into the account.
   - withdraw(double amount): Withdraws the specified amount from the account. If the withdrawal amount is greater than the balance, print an insufficient funds message.
   - getBalance(): Returns the current balance.
   - displayAccountInfo(): Prints the account details (account number, holder name, and balance).
5. **Bank Class (Optional - for more advanced students):** Create a Bank class that can hold an array of Account objects. Implement methods in the Bank class to add new accounts, search for accounts, and perform transactions.
6. **Test:** Create a test class to create accounts, perform deposits and withdrawals, and display account information.

Discussion and analysis of results:

Conclusion:

Lab Session 4

Objective:

Understanding the concepts of **method overloading**, **constructor overloading**, and **access control** in Java.

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:

1. 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);
   }
}
```

2. **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());
    }
}
```

Observation:

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: Course and 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.

1. **Home Task #1**

Overloading with Different Data Types

In the OverloadDemo class, we have a method named test that is overloaded with different parameter types (no parameters, int, double, and two int parameters). Imagine that you are developing a calculator application that handles different mathematical operations. You are tasked with adding new functionality to the test method to support additional operations for complex numbers.

Exercise:

- Add a new method test that accepts two Complex number objects as parameters (you can create a simple Complex class with real and imaginary parts). The method should print the sum of the two complex numbers.

- Define the Complex class with appropriate constructors, getters, and setters for real and imaginary parts. Also, implement a toString() method to display a complex number as a string in the format "real + imaginary i".

2. Home Task #2

Constructor Overloading with Default and Parameterized Constructors

You are developing a banking system and need to handle different account types (e.g., Savings, Checking). The system should allow creation of accounts either with a default ID and balance, or with a specific ID and balance.

Exercise:

- Modify the Account class to add a constructor for a new type of account, called CheckingAccount, that accepts an additional parameter: isOverdraftAllowed (boolean). This boolean indicates whether overdraft is allowed for the account.

- Ensure that the CheckingAccount constructor also sets the ID, balance, and initializes dateCreated. Add getter and setter methods for the isOverdraftAllowed property.

3. Home Task #3

Access Control in Banking System

In a banking application, it’s crucial to ensure that certain details of an account (such as balance) are not directly accessible by unauthorized users. For this, you will utilize Java’s access control modifiers (private, protected, public).

Exercise:

- Implement a Customer class with a private field balance and a public method getBalance to retrieve the balance. In addition, provide a protected method setBalance to modify the balance (to be used only by subclasses or within the same package).
- Create a subclass VIPCustomer that extends the Customer class. In VIPCustomer, override the setBalance method to allow balance modification for VIP customers without restrictions (i.e., no negative balance).
- Demonstrate the usage of access modifiers by attempting to access the balance field and the getBalance and setBalance methods from a different class, showing what is allowed and what is not.

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:

1. **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);
   }
}
```

2. **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();
      }
   }
}
```

3. **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));
    }
}
```

Observation:

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.

1. Home Task #1

Finding the Second Largest Element in an Array

You are tasked with developing a software application for a retail company that processes sales data. Each day, the company tracks the sales amounts for different products. The company needs a feature to identify the second highest sales figure from a list of daily sales figures for the week. This will help them recognize the second best-performing product.

Exercise:

- Write a Java program that takes an array of sales figures (integers) for the week (7 days).
- The program should find and print the second highest sales figure from the array.
- The program must handle the case where there are duplicate sales values and where the array has fewer than two distinct sales figures.

Requirements:

1. Implement a method findSecondLargest(int[] sales) that returns the second largest number in the array.
2. The program should check if there are at least two distinct sales values; if not, it should notify the user that the second largest value cannot be found.
3. Consider edge cases such as an array with one element, an array with all the same values, and an array with two elements where both are the same.

2. Home Task #2

Finding the Second Smallest Element in an Array

You are building an inventory management system for an e-commerce company. The system keeps track of the stock levels for various products in a warehouse. Each product has a corresponding stock quantity, and the company needs a way to identify the second lowest stock level in order to restock products that are running low.

Exercise:

- Write a Java program that takes an array of integers representing the stock levels of different products in the warehouse.
- The program should find and print the second smallest stock quantity in the array.
- The program should handle edge cases such as an array with duplicate stock quantities or arrays with fewer than two distinct stock values.

Requirements:

1. Implement a method findSecondSmallest(int[] stock) that returns the second smallest element in the array.
2. The program should check if there are at least two distinct stock values; if not, it should notify the user that the second smallest stock quantity cannot be determined.
3. Consider edge cases such as arrays with a single element, arrays where all elements are the same, and arrays with only two elements where both have the same value.

3. Home Task #3

Managing a Library System with Books Using Arrays

You are developing a Library Management System for a public library. The system needs to keep track of books in the library, their titles, authors, and the number of available copies. The library can store information about multiple books, and users should be able to query the library for book details, add new books, and check the total number of books available.

Exercise:

- Create a class Book that has the following properties:

- title (String)
- author (String)
- copiesAvailable (int)

Include the following methods in the Book class:

- A constructor to initialize the properties.
- A method getBookInfo() that returns a string with the book's title, author, and the number of available copies.
- A method addCopies(int newCopies) that adds the specified number of new copies to the copiesAvailable field.
- A method borrowBook() that decreases the copiesAvailable by 1 (if available).

- Create another class Library that will manage an array of Book objects. The Library class should have the following functionalities:

- A method addBook(Book book) that adds a new book to the library's collection.
- A method findBookByTitle(String title) that returns a Book object based on the title provided. If no book is found, it should return a message saying "Book not found."
- A method totalBooks() that returns the total number of books in the library.
- A method borrowBookFromLibrary(String title) that allows a user to borrow a book by title. It should display a message saying whether the borrowing was successful (if the book is available) or if the book is out of stock.

Discussion and analysis of results:

Conclusion:

Lab Session 6

Objective:

Understanding inheritance and the relationships between superclasses and subclasses in Java.

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:

1. **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);
   }
}
```

Observations:

2. **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);
    }
}
```

Observations:

3. 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());
    }
}
```

Observations:

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.

1. **Home Task #1**

Scenario 1: Employee Management System

You are tasked with developing an Employee Management System for a company that has different types of employees such as full-time employees, part-time employees, and contract employees. All employees share common attributes, such as `name`, `id`, and `salary`, but their salary calculations differ based on employment type.

Exercise:

- Create a base class `Employee` with the following properties:
  - `name` (String)
  - `id` (int)
  - `salary` (double)

  Include the following methods in the `Employee` class:

  - A constructor to initialize the properties.
  - A method `calculateSalary()` that returns the salary of the employee.
- Create three subclasses: `FullTimeEmployee`, `PartTimeEmployee`, and `ContractEmployee`. Each subclass should:
  - Override the `calculateSalary()` method with specific rules:
    - `FullTimeEmployee`: Fixed monthly salary.
    - `PartTimeEmployee`: Hourly wage multiplied by the number of hours worked.
    - `ContractEmployee`: A daily rate multiplied by the number of days worked.
  - Implement their own constructor to initialize the fields with different parameters as needed.

2. **Home Task #2**

Vehicle Insurance System

You are designing a vehicle insurance system that handles different types of vehicles such as cars, motorcycles, and trucks. All vehicles have some common properties, such as `make`, `model`, and `year`, but the insurance calculations differ based on the type of vehicle.

Exercise:

- Create a base class `Vehicle` with the following properties:
  - `make` (String)
  - `model` (String)
  - `year` (int)

  Include the following methods:

  - A constructor to initialize the properties.
  - A method `calculateInsurance()` that returns the base insurance cost for the vehicle.
- Create three subclasses: `Car`, `Motorcycle`, and `Truck`. Each subclass should:
  - Override the `calculateInsurance()` method with specific rules:
    - `Car`: Base cost plus an additional percentage based on the car's value.
    - `Motorcycle`: Lower base cost with a higher rate for high-performance motorcycles.
    - `Truck`: Higher base cost with additional fees based on the truck's weight and cargo capacity.
  - Implement their own constructors to initialize the vehicle type-specific attributes.

3. **Home Task #3**

Banking System

You are building a banking system where customers can have different types of bank accounts. Each account type has its own rules for withdrawing money and applying interest, but all accounts share some common properties like `accountNumber`, `balance`, and `ownerName`.

Exercise:

- Create a base class `BankAccount` with the following properties:

    - `accountNumber` (String)
    - `balance` (double)
    - `ownerName` (String)

  Include the following methods:

    - A constructor to initialize the properties.
    - A method `deposit(double amount)` that adds funds to the account.
    - A method `withdraw(double amount)` that subtracts funds from the account (ensure balance cannot go below zero).
    - A method `applyInterest()` that calculates and adds interest to the balance based on a fixed rate.

- Create three subclasses: `SavingsAccount`, `CheckingAccount`, and `BusinessAccount`. Each subclass should:

    - `SavingsAccount`: Apply a fixed annual interest rate to the balance.
    - `CheckingAccount`: No interest, but offer an overdraft protection feature.
    - `BusinessAccount`: Apply a different interest rate and allow higher withdrawal limits.

Discussion and analysis of results:

Conclusion:

Lab Session (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 7

Objective:

Understanding the abstract methods & classes, and final methods and classes.

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

a) **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);
}
```

b) **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
    }
}
```

c) **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());
    }
}
```

2. Final Methods and Classes

a) **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.");
    }
}
```

b) **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.");
    }
}
```

Observations:

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.

1. **Home Task #1**

Abstract Class for Shapes

1. Create an abstract class `Shape` with:
   - An abstract method `calculateArea()`.
   - A non-abstract method `displayArea(double area)` to display the calculated area.
2. Extend `Shape` with concrete classes `Circle` and `Rectangle`:
   - `Circle` should calculate the area using π×radius2\pi \times \text{radius}^2.
   - `Rectangle` should calculate the area using length×breadth\text{length} \times \text{breadth}.
3. Write a `main` method to create objects of `Circle` and `Rectangle`, call their `calculateArea()` methods, and display the results.

2. **Home Task #2**

Task 2: Abstract Class for Employees

1. Define an abstract class `Employee` with:
   - Abstract methods `calculateSalary()` and `getRole()`.
   - Non-abstract attributes for `name` and `employeeID` and a method `displayDetails()`.
2. Create concrete subclasses `Manager` and `Developer`:
   - `Manager` should have a bonus-based salary calculation.
   - `Developer` should calculate salary based on hours worked.
3. Write a program to display the details of both types of employees.

3. **Home Task #3**

Task 3: Abstract Class for Appliances

1. Create an abstract class `Appliance` with:
   - Abstract methods `turnOn()` and `turnOff()`.
   - A non-abstract method `displayStatus(String status)`.
2. Extend `Appliance` with concrete classes `Fan` and `WashingMachine`:
   - `Fan` should have specific logic for turning on and off.
   - `WashingMachine` should include additional functionality for starting/stopping wash cycles.
3. Demonstrate the functionality of these classes in a `main` method.

Discussion and analysis of results:

Conclusion:

Lab Session 8

Objective:

Understanding the concept of packages & interfaces of Java.

Required Equipment / tools:

- Eclipse
- JDK (Java Development Kit)

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
   ```

3. 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.

Observations:

Task #1

Scenario: You are developing an application that simulates a library system. The system should handle various operations like borrowing and returning books. The application needs to allow users to interact with different types of books (e.g., Fiction, Non-Fiction, etc.) using common methods defined by an interface. You are also required to organize the code into different packages for better structure and modularity.

Task Description:

1. **Create an interface: Book** with methods for `borrow()` and `returnBook()`.
2. **Implement classes** for different types of books (e.g., `FictionBook`, `NonFictionBook`, etc.) that implement the `Book` interface. Each class should provide specific implementations for the `borrow()` and `returnBook()` methods.
3. **Create a package** `library` that contains the `Book` interface and the implementing classes (like `FictionBook` and `NonFictionBook`).
4. **Create another package** `librarymanagement` that includes a main class to simulate user interactions with the library system, such as borrowing and returning books.
5. In the main class, **use the Book interface** to interact with different book types, ensuring flexibility and adherence to the principles of object-oriented programming.

Task #2

Scenario: A software company is developing a **Home Automation System** that allows users to control various smart devices like lights, thermostats, and security cameras. The company wants to ensure that the system can easily accommodate new types of smart devices in the future. They decide to use interfaces and packages to create a flexible and modular design, allowing each smart device to implement a common interface.

Task Description:

1. **Define the `SmartDevice` interface** with methods for `turnOn()` and `turnOff()`. The `turnOn()` method will activate the device, and the `turnOff()` method will deactivate it.
2. **Create classes for different types of smart devices** such as `Light`, `Thermostat`, and `SecurityCamera` that implement the `SmartDevice` interface. Each class should provide specific implementations for the `turnOn()` and `turnOff()` methods based on the type of device.
3. **Create a package** called `smarthome` that contains the `SmartDevice` interface and the implementing classes (e.g., `Light`, `Thermostat`, and `SecurityCamera`).
4. **Create another package** called `homeautomation` that includes a main class to simulate user interactions, such as turning on and off the devices. The main class should use the `SmartDevice` interface to interact with different types of devices.

Task #3

Scenario: In a **media player system**, you need to manage various types of media files like audio files, video files, and images. Each media type has specific operations but should also adhere to common media functionalities.

Task Description:

1. **Create a package** named `media` to contain all media-related classes.
2. **Define two interfaces**:
   - `Playable` with methods such as `play()` and `pause()`.
   - `Viewable` with methods such as `zoomIn()` and `zoomOut()`.
3. **Implement classes** for different media types like `Audio`, `Video`, and `Image` that implement the `Playable` interface. The `Video` and `Image` classes should also implement the `Viewable` interface. Each class should define its unique attributes and methods while adhering to the interface contracts.
4. **Create a main class** in a package called `mainpkg` and implement all the media classes in the main class.
5. **Show the result**.

1. **Home Task #1**

Create and Use Packages

1. Create a package named `shapes` that contains a class `Circle` with a method to calculate the area of a circle.
2. Create another package named `operations` that contains a class `Calculator` with methods for addition and subtraction.
3. Write a `main` class in a separate package to use classes from both packages, demonstrating `import` and `fully qualified names`.

2. **Home Task #2**

Interface Implementation

1. Define an interface `Vehicle` with methods `startEngine()` and `stopEngine()`.
2. Create two classes, `Car` and `Bike`, that implement the `Vehicle` interface.
3. Write a `main` method to create objects of `Car` and `Bike` and invoke their methods.

3. **Home Task #3**

Interface as a Callback

1. Define an interface `EventListener` with a method `onClick(String eventSource)`.
2. Create a class `Button` that accepts an `EventListener` and simulates an event (e.g., button click).
3. Write a class `UserAction` that implements `EventListener` and handles the `onClick` event.

Lab Session 9

Objective:

Understanding how runtime errors are managed in Java using exception handling mechanisms.

Required Equipment / tools:

- Eclipse
- JDK (Java Development Kit)

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.

4. 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:

a) Enter non-numeric input to trigger the exception.
b) Verify that the exception is handled, and the program terminates gracefully.

2. 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());
        }
    }
```

Observations:

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, and Truck 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.

1. **Home Task #1**

Custom Exception

1. Create a custom exception `InsufficientFundsException` for a bank application.
2. Write a `BankAccount` class with a method `withdraw(double amount)` that throws `InsufficientFundsException` if the withdrawal amount exceeds the balance.
3. Write a `main` method to demonstrate the usage of this custom exception.

2. **Home Task #2**

Handling Multiple Exceptions

1. Write a program that accepts two numbers from the user and performs division.
2. Handle exceptions for:
   - Division by zero (`ArithmeticException`)
   - Invalid input (`NumberFormatException`)
3. Use a `try-catch` block to catch and display meaningful error messages.

3. **Home Task #3**

Try-With-Resources

1. Create a file handling program that reads data from a file using a `FileReader`.
2. Implement exception handling using a `try-with-resources` statement to automatically close the file.
3. Handle `FileNotFoundException` and `IOException`.

Discussion and analysis of results:

Conclusion:

Lab Session 10

Objective:

Understanding GUI design principles by exploring JavaFX components such as Labels, Buttons, Text Boxes, Combo Boxes, and layouts, focusing on their interactions and applications in creating user-friendly and interactive interfaces.

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;
```

2. **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
    }
}
```

3. **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");
```

4. **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);
```

5. **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();
```

Observations:

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:

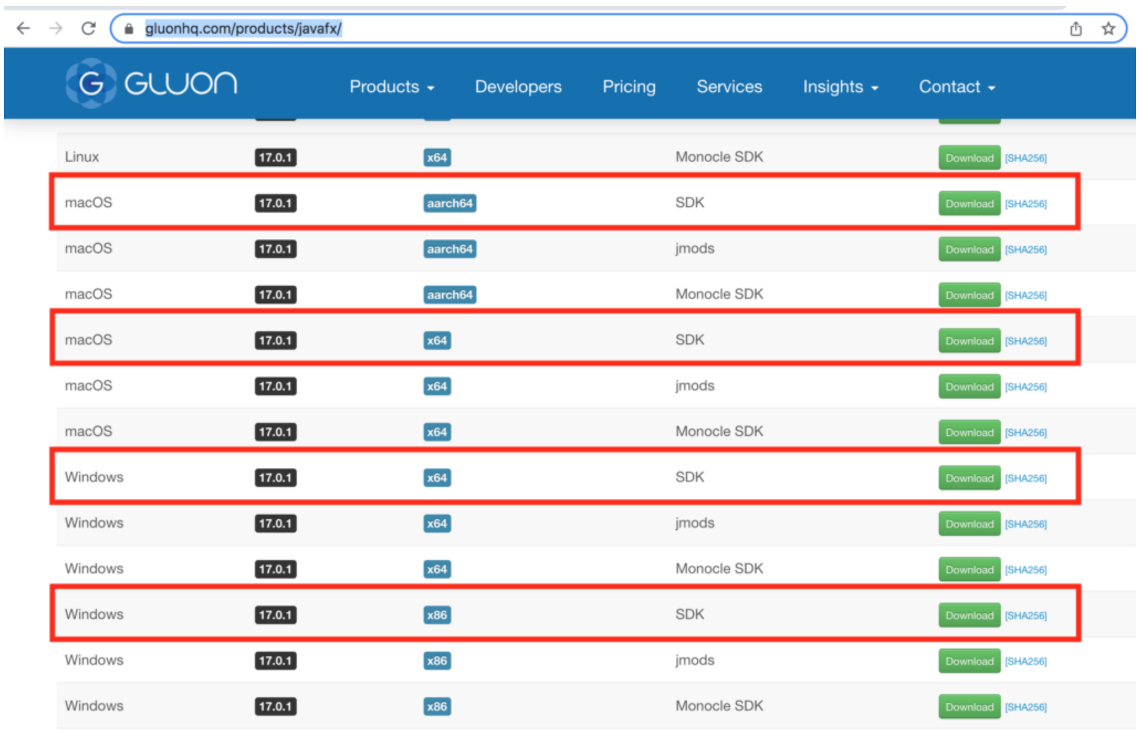
Lab Session 10

Objective: Objective: Introducing JavaFX – Java GUI

Introduction:

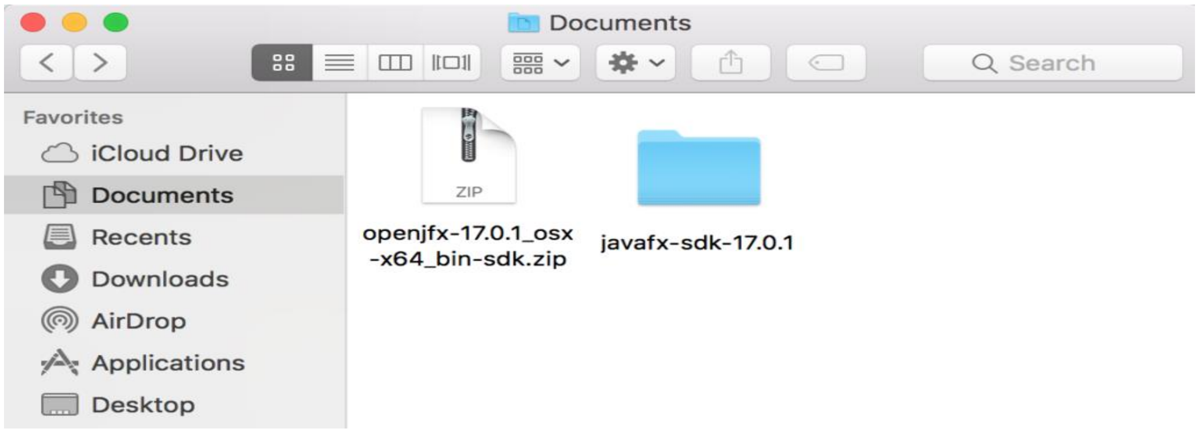
STEP 1. DOWNLOAD JAVAFX

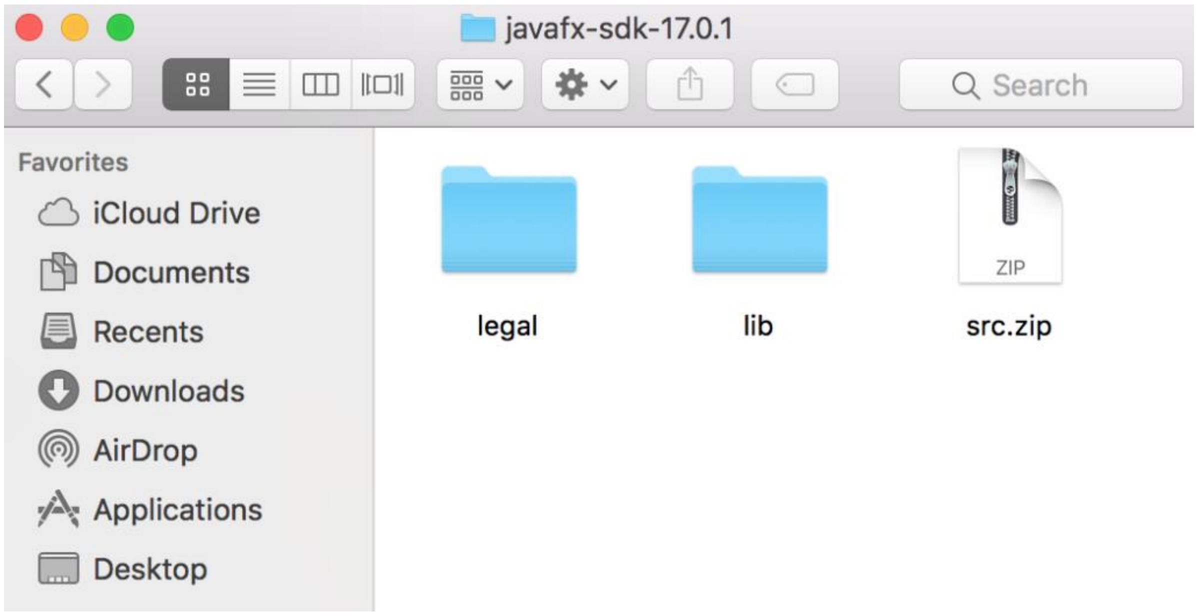
Go to https://gluonhq.com/products/javafx/ and download the appropriate **SDK** for your operating system. Make sure you download the SDK and make sure you choose the correct operating system and architecture for your computer. More than likely, you'll be downloading one of the outlined options below.



Linux	17.0.1	x64	Monocle SDK	Download [SHA256]
macOS	17.0.1	aarch64	SDK	Download [SHA256]
macOS	17.0.1	aarch64	jmods	Download [SHA256]
macOS	17.0.1	aarch64	Monocle SDK	Download [SHA256]
macOS	17.0.1	x64	SDK	Download [SHA256]
macOS	17.0.1	x64	jmods	Download [SHA256]
macOS	17.0.1	x64	Monocle SDK	Download [SHA256]
Windows	17.0.1	x64	SDK	Download [SHA256]
Windows	17.0.1	x64	jmods	Download [SHA256]
Windows	17.0.1	x64	Monocle SDK	Download [SHA256]
Windows	17.0.1	x86	SDK	Download [SHA256]
Windows	17.0.1	x86	jmods	Download [SHA256]
Windows	17.0.1	x86	Monocle SDK	Download [SHA256]

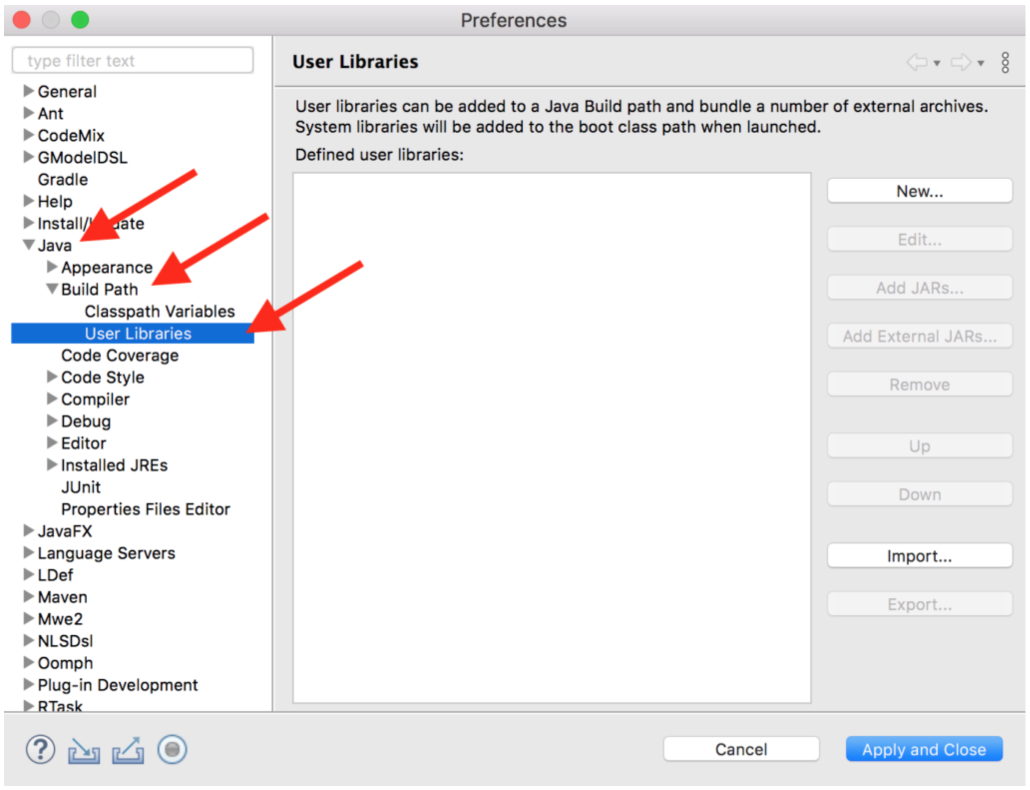
This will install a `.zip` file. Unzip this file and put the resulting folder in a memorable place, such as your `/Documents` folder. We'll need this later.

- 

- 

STEP 2. CREATE A USER LIBRARY IN ECLIPSE

- Open up Eclipse and go to **Preferences**
- You should be able to get to Preferences from Eclipse > Preferences or Window > Preferences
- Go to **Java** >**Build Path** > **User Libraries**

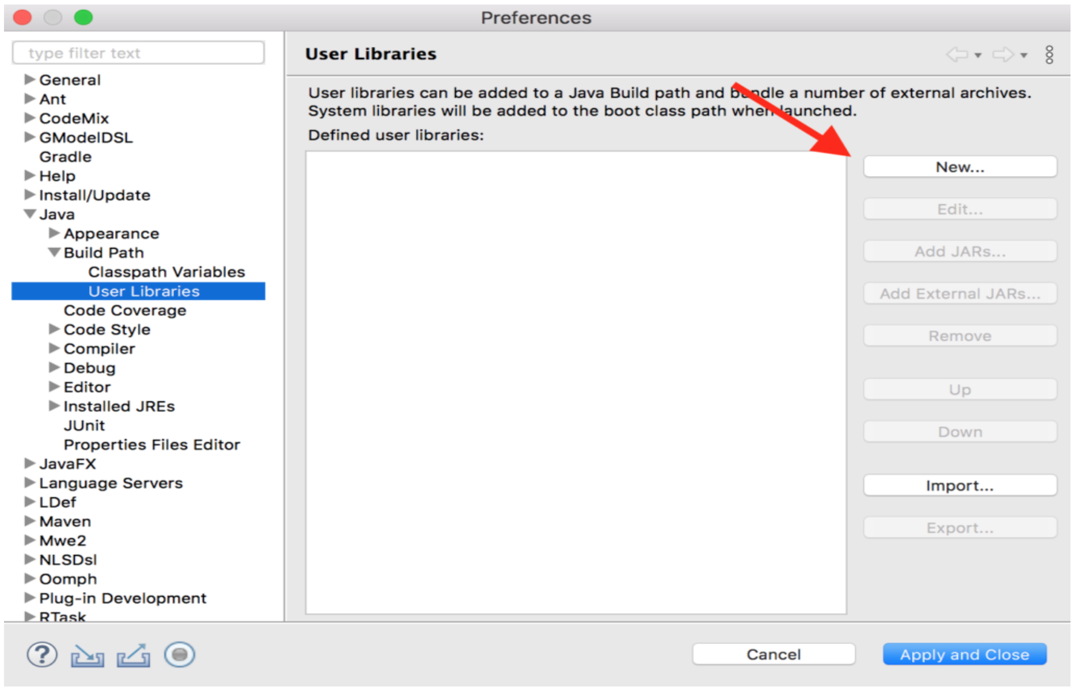


User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

- Now click **New…**

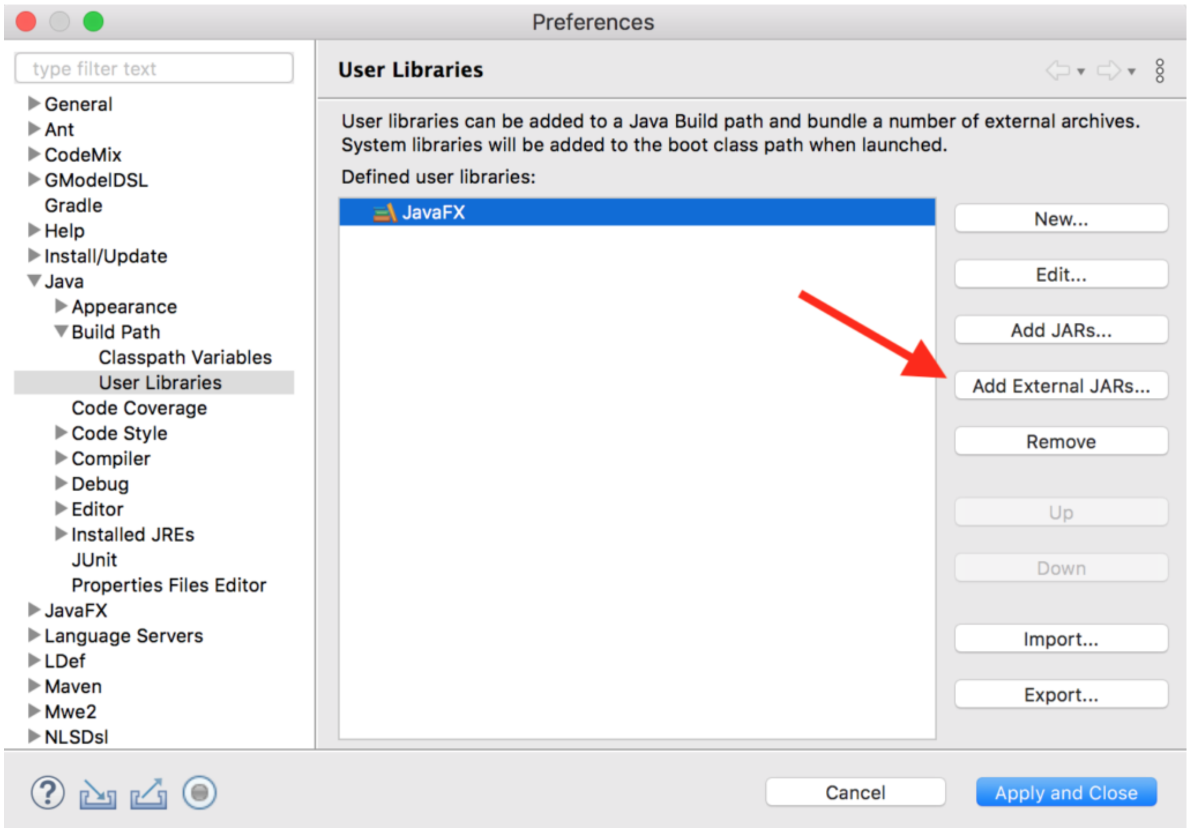


User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

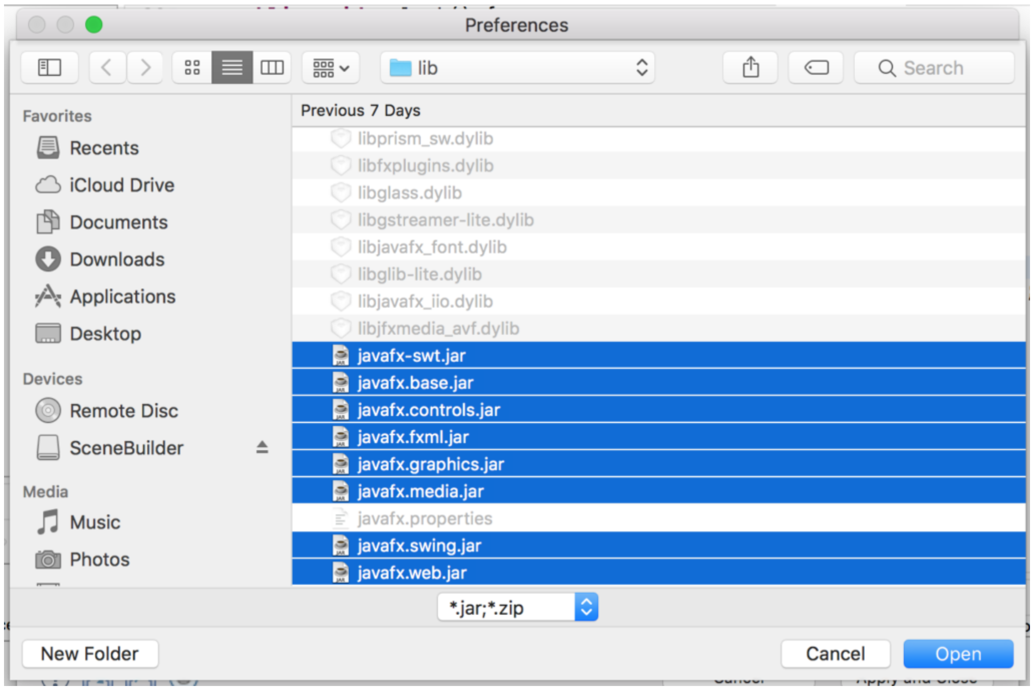
Defined user libraries:

- Name it **JavaFX** and click **Ok**
- With JavaFX highlighted, click **Add External JARs…**

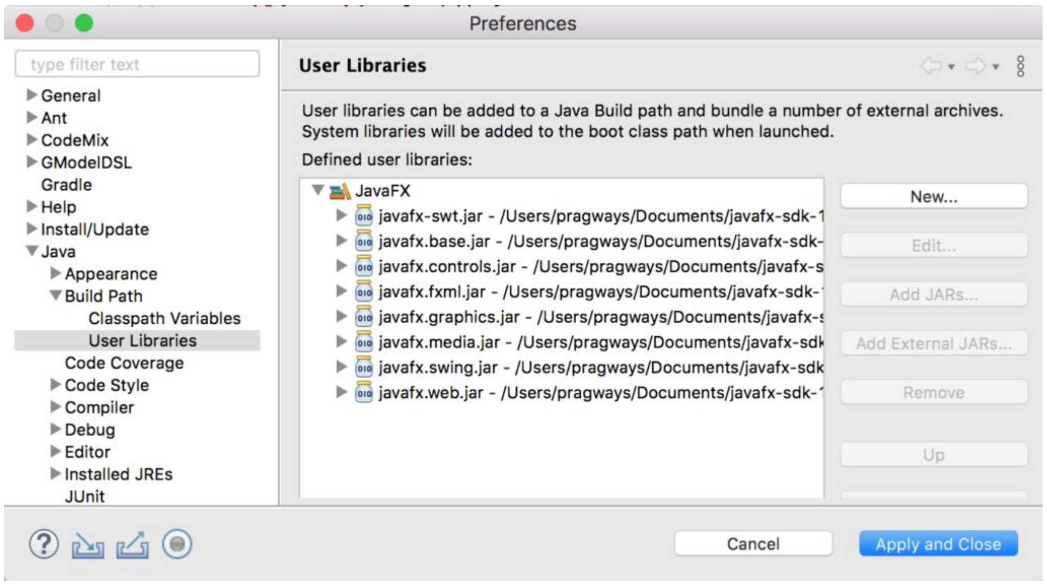


User libraries can be added to a Java Build path and bundle a number of external archives.
System libraries will be added to the boot class path when launched.
Defined user libraries:

- Select all the `.jar` files from the `/lib/` folder of the unzipped JavaFX folder we saved earlier.



- Click **Open** and your new User Library should look something like this:



User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives.
System libraries will be added to the boot class path when launched.

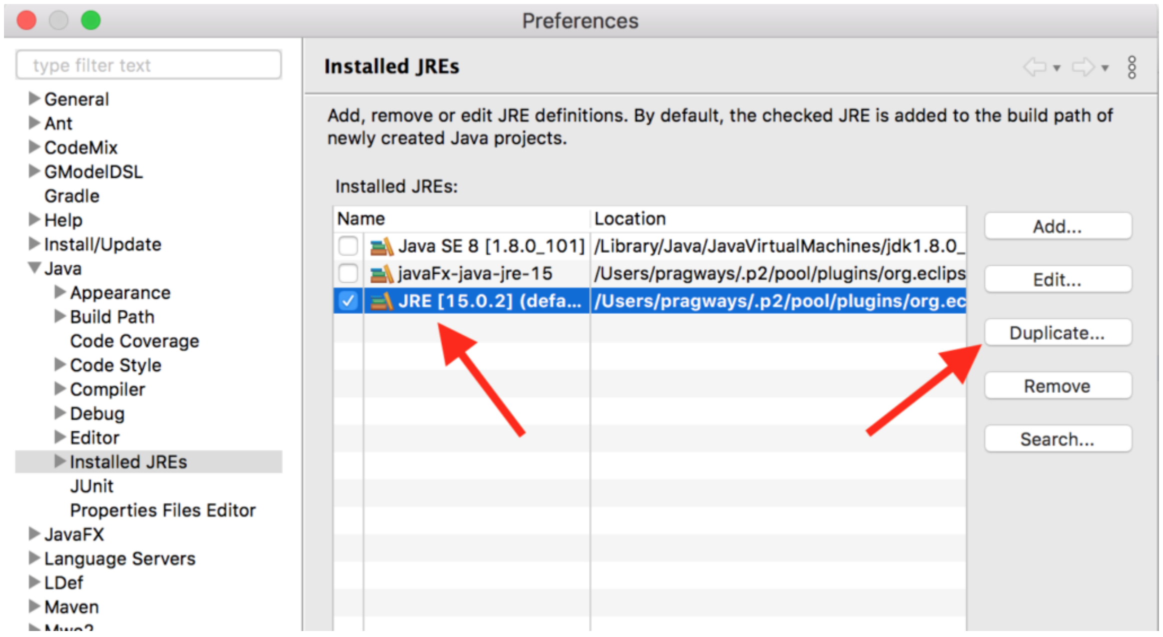
Defined user libraries:

```
JavaFX
  javafx-swt.jar - /Users/pragways/Documents/javafx-sdk-1
  javafx.base.jar - /Users/pragways/Documents/javafx-sdk-
  javafx.controls.jar - /Users/pragways/Documents/javafx-s
  javafx.fxml.jar - /Users/pragways/Documents/javafx-sdk-
  javafx.graphics.jar - /Users/pragways/Documents/javafx-s
  javafx.media.jar - /Users/pragways/Documents/javafx-sdk
  javafx.swing.jar - /Users/pragways/Documents/javafx-sdk
  javafx.web.jar - /Users/pragways/Documents/javafx-sdk-1
```

- Click **Apply and Close** to save your new User Library

Step 3. Create a JRE clone with the required VM arguments

- Go back to **Preferences** in Eclipse
- Go to **Java** > **Installed JREs**
- Select your default JRE and click **Duplicate**



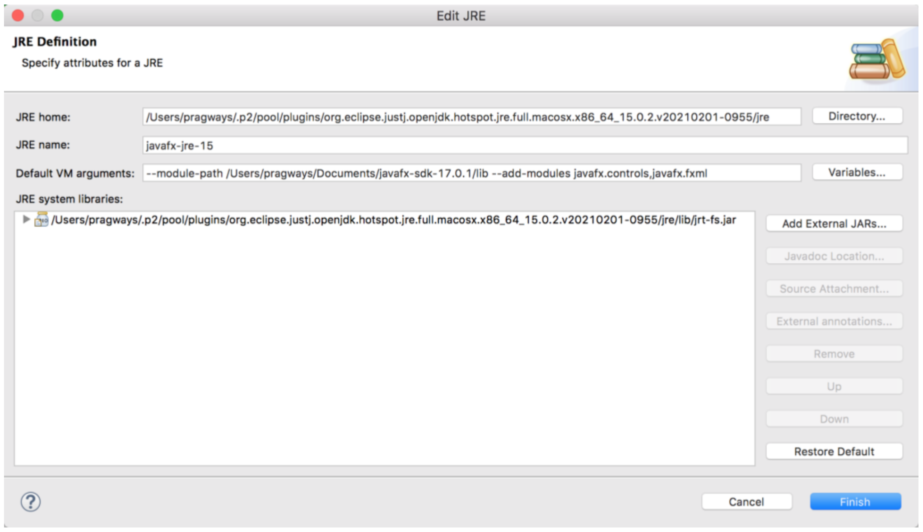
Installed JREs

Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

	Name	Location
☐	Java SE 8 [1.8.0_101]	/Library/Java/JavaVirtualMachines/jdk1.8.0_
☐	javaFx-java-jre-15	/Users/pragways/.p2/pool/plugins/org.eclips
✓	JRE [15.0.2] (defa...	/Users/pragways/.p2/pool/plugins/org.ec

- Copy this line if you’re on **Windows**
- `--module-path "\path\to\javafx-sdk-17\lib" --add-modules javafx.controls,javafx.fxml`
- Copy this line if you’re on **Mac** or **Linux**
- `--module-path /path/to/javafx-sdk-17/lib --add-modules javafx.controls,javafx.fxml`
- **Important:** Make sure you replace the `/path/to/javafx-sdk-17/lib` with the path to where you placed your unzipped JavaFX folder from before. You will want the entire path all the way to the /lib/ folder. For instance, mine on a Mac looks like this:
- `--module-path /Users/pragways/Documents/javafx-sdk-17.0.1/lib --add-modules javafx.controls,javafx.fxml`
- Paste that line in the **Default VM arguments:** field
- Rename the **JRE name:** field to something memorable, such as **javafx-jre-15**



JRE home: /Users/pragways/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.2.v20210201-0955/jre

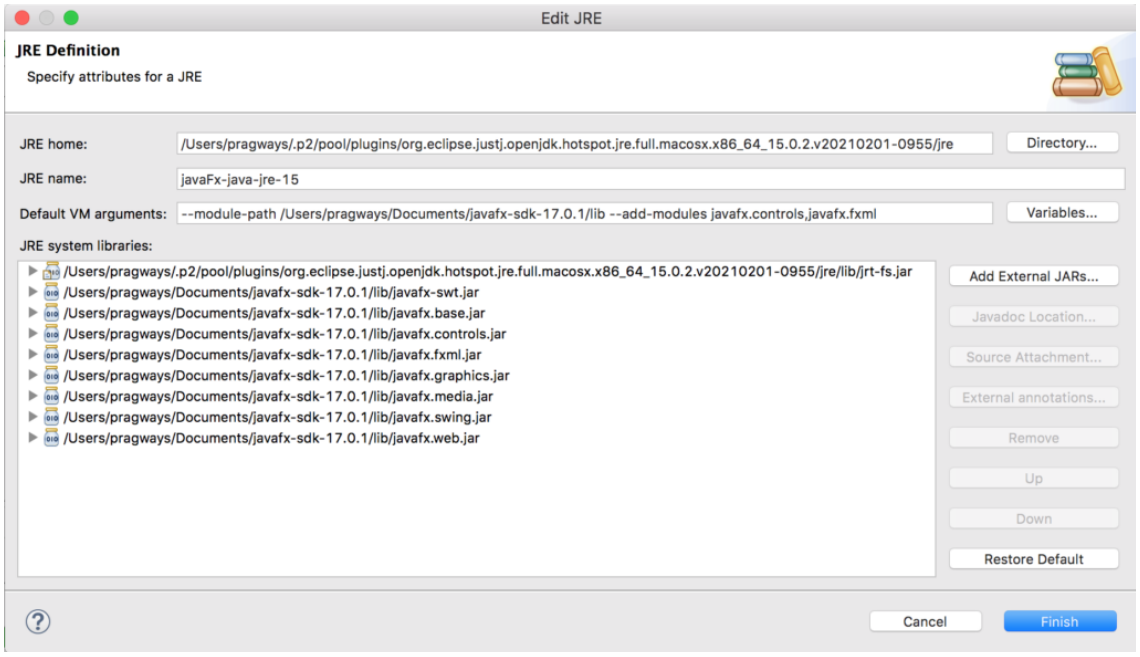
JRE name: javafx-jre-15

Default VM arguments: --module-path /Users/pragways/Documents/javafx-sdk-17.0.1/lib --add-modules javafx.controls,javafx.fxml

JRE system libraries:

```
/Users/pragways/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.2.v20210201-0955/jre/lib/jrt-fs.jar
```

- Click **Add External JARs…**
- Select all the `.jar` files from the `/lib/` folder of the unzipped JavaFX folder we saved earlier and then click **Open**



JRE home: /Users/pragways/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.2.v20210201-0955/jre

JRE name: javaFx-java-jre-15

Default VM arguments: --module-path /Users/pragways/Documents/javafx-sdk-17.0.1/lib --add-modules javafx.controls,javafx.fxml

JRE system libraries:

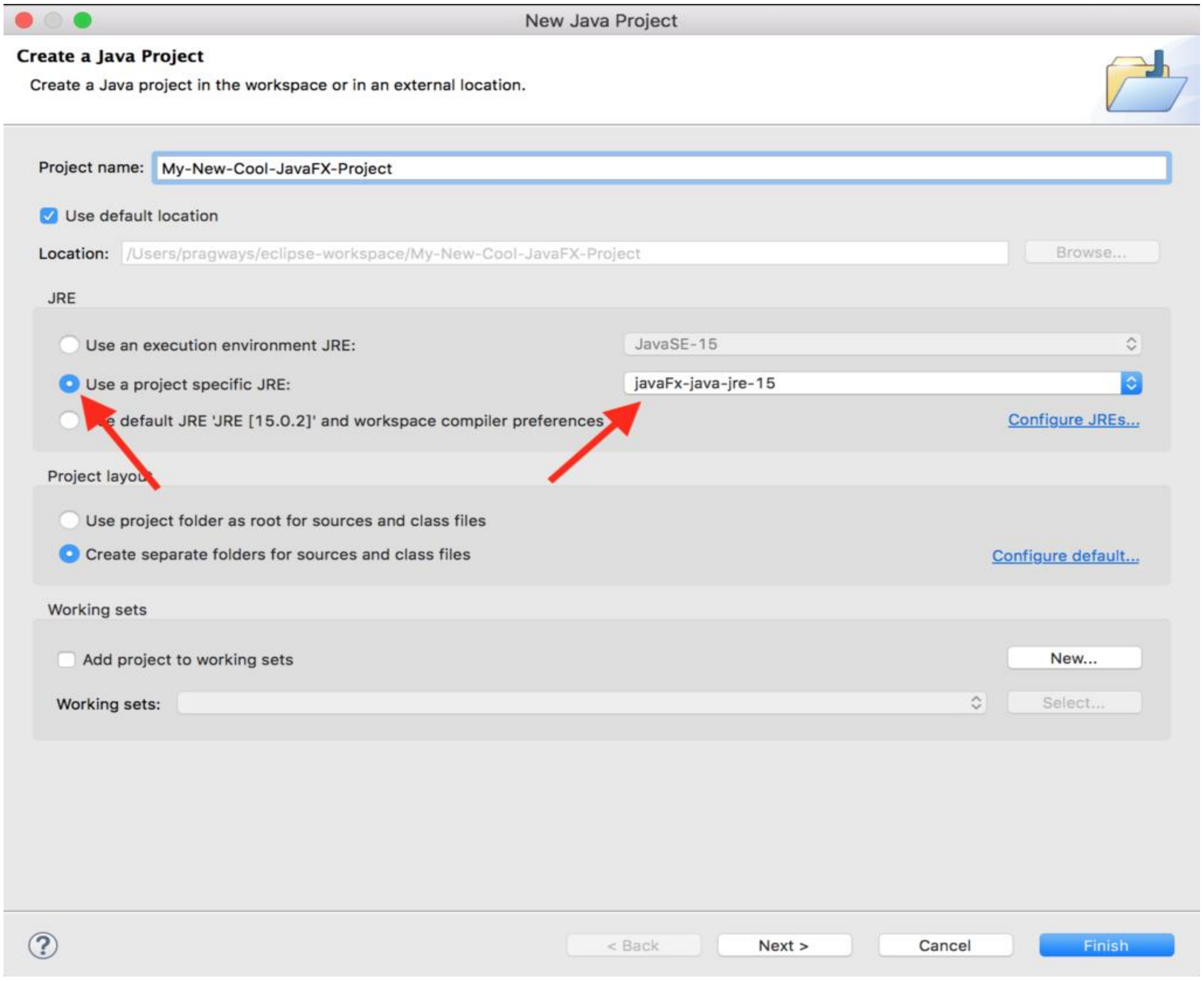
```
/Users/pragways/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.2.v20210201-0955/jre/lib/jrt-fs.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx-swt.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.base.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.controls.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.fxml.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.graphics.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.media.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.swing.jar
/Users/pragways/Documents/javafx-sdk-17.0.1/lib/javafx.web.jar
```

- Your JRE Definition window should now look something like this
- Click **Finish**

STEP 4. CREATE A NEW PROJECT WITH YOUR NEW JRE

Now whenever you want to create a new JavaFX project, it will be a lot quicker and easier!

All you need to do is make sure you select **Use a project specific JRE** when creating a new project, and then select your new JavaFX specific JRE that you just created.


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: My-New-Cool-JavaFX-Project

[✓] Use default location

Location: /Users/pragways/eclipse-workspace/My-New-Cool-JavaFX-Project

JRE

( ) Use an execution environment JRE: JavaSE-15

(●) Use a project specific JRE: javaFx-java-jre-15

( ) Use default JRE 'JRE [15.0.2]' and workspace compiler preferences

Configure JREs...

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Configure default...

Working sets

[ ] Add project to working sets

Working sets:

Procedure:

```
import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.StackPane;

import javafx.stage.Stage;


public class HelloJavaFX extends Application {

    @Override

    public void start(Stage primaryStage) {

        // Create a Button

        Button button = new Button("Click Me!");

        button.setOnAction(e -> System.out.println("Hello, JavaFX!"));


        // Layout and Scene

        StackPane layout = new StackPane();

        layout.getChildren().add(button);

        Scene scene = new Scene(layout, 300, 200);


        // Stage setup

        primaryStage.setTitle("Hello JavaFX");

        primaryStage.setScene(scene);

        primaryStage.show();

    }
    public static void main(String[] args) {
        launch(args); // Launch JavaFX application
    }
}
```

Expected Outcome:

Task 1: You are tasked with creating a simple JavaFX application that displays a personalized greeting message when a user enters their name and clicks a button.

A user opens the application and sees:

1. A **TextField** where they can type their name.
2. A **Button** labeled "Say Hello".
3. A **Label** that displays a greeting message.

When the user enters their name (e.g., "Alice") in the text field and clicks the "Say Hello" button, the label updates to display "Hello, Alice!".

Requirements

1. Use JavaFX's `TextField`, `Button`, and `Label` components.
2. Implement an **event handler** for the button click to update the label with the greeting message.
3. Initialize the application with an empty text field and a label that displays a default message like "Welcome!".

Task 2: You are tasked with creating a JavaFX application that changes the background color of the window when the user clicks a button.

The application starts with:

1. A window with a default white background.
2. A **Button** labeled "Change Color".

When the user clicks the "Change Color" button:

- The background color of the window changes to a random color.

Requirements

1. Use JavaFX's `Button` and `Pane` components.
2. Generate a random color using JavaFX's `Color` class.
3. Change the background color of the window's `Pane` each time the button is clicked.

Task 3:

You are tasked with creating a JavaFX application that increments a counter every time a button is clicked.

The application starts with:

1. A **Label** displaying the number `0` (the counter's initial value).
2. A **Button** labeled "Increment".

Each time the user clicks the "Increment" button:

- The number displayed in the label increases by 1.

Requirements

1. Use JavaFX `Label` and `Button` components.
2. Update the label's text dynamically based on button clicks.
3. Keep the layout simple, with the label above the button.

Home Task 1: Create a simple JavaFX application to convert temperatures between Celsius and Fahrenheit.
Task Description:

1. **Layout:**
   - Add a Label for the title (e.g., "Temperature Converter").
   - Add two TextField components for entering the temperature in Celsius and Fahrenheit.
   - Include two Buttons labeled "Convert to Fahrenheit" and "Convert to Celsius".
   - Arrange the components using a VBox.
2. **Functionality:**
   - Clicking "Convert to Fahrenheit" should calculate the Fahrenheit equivalent and display it in the appropriate TextField.
   - Clicking "Convert to Celsius" should calculate the Celsius equivalent and display it.
3. **Validation:**
   - Display an error message in a Label if the user enters invalid input.
4. **Enhancement:**
   - Add tooltips to guide users on how to use the application.

Home Task 2: To-Do List Manager

Build a basic To-Do List application.
Task Description:

1. **Layout:**
   - Add a TextField for entering a task.
   - Include an "Add Task" button to add the task to a list.
   - Use a ListView to display the tasks.
   - Add a "Remove Selected Task" button to delete a selected task.
   - Arrange components using a VBox.
2. **Functionality:**
   - Clicking "Add Task" should add the entered task to the ListView.
   - Clicking "Remove Selected Task" should delete the selected task from the ListView.
3. **Enhancement:**
   - Add a confirmation dialog (Alert) before deleting a task.

Home Task 3: Simple Calculator

Scenario: Create a basic calculator application to perform addition, subtraction, multiplication, and division.

Task Description:

1. **Layout:**
    - Add two TextField components for number input.
    - Include four Buttons for "+", "-", "*", and "/".
    - Add a Label to display the result.
    - Use an HBox for the buttons and a VBox for the overall layout.
2. **Functionality:**
    - Clicking a button should perform the corresponding operation and display the result in the Label.
    - Handle division by zero with an error message in the Label.
3. **Validation:**
    - Ensure both inputs are valid numbers before performing any operation.

Lab Session 11

Objective:

Understanding and implementing various layouts and charts in JavaFX to create enhanced and interactive 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.

Home Task 1: Create a Dynamic File Management System

Description: Develop a JavaFX application to manage and preview files stored on a local directory. The application should allow users to view, search, and organize files dynamically.

Instructions:

1. Design a BorderPane layout with:
   - **Top**: A label titled "File Management System" and a TextField search bar to filter files by name.
   - **Left**: A vertical tree structure (TreeView) showing directories and subdirectories from a local folder.
   - **Center**: A table displaying file details such as "File Name," "Size," and "Last Modified".
   - **Right**: A preview pane showing the content of the selected file (for text files).
   - **Bottom**: A footer displaying the total number of files and their combined size.
2. Implement the following functionality:
   - Clicking on a directory in the TreeView should load its files into the table.
   - Use the search bar to filter files in the table based on their names.
   - Display a file's content in the preview pane when clicked.
3. Save the current directory and search state when the application is closed and reload it on restart.

Home Task 2: Task: Create a Library Catalog System with Interactive Book Details

Description: Develop a JavaFX application to manage a library catalog, allowing users to browse and view details of books interactively.

Instructions:

1. Use a TilePane to display a grid of book covers represented by buttons with titles (e.g., "Book 1," "Book 2").
2. Clicking a button should open a new window (Stage) displaying:
   - The book’s title.
   - Author’s name.
   - Genre.
   - A short description of the book.
3. Add a "Mark as Favorite" button in the new window. When clicked, store the book in a separate "Favorites" list.
4. Include a "View Favorites" button in the main window to display a list of all marked favorite books in a separate scene.
5. Ensure the application can save the favorites list to a file and reload it when reopened

Home Task 3: Develop a Survey Application with Real-Time Results Visualization

Description: Build a JavaFX application to conduct a survey and display real-time survey results using a GridPane for questions and a PieChart for visualization.

Instructions:

1. Use a GridPane to display a survey with the following:
   - Multiple questions (e.g., "What is your favorite color?").
   - Each question should have options represented as RadioButton groups (e.g., "Red," "Blue," "Green").
2. Include a "Submit Survey" button at the bottom.
3. Upon submission:
   - Calculate and display the count of responses for each option in real-time.
   - Use a PieChart to visualize the percentage of responses for each option dynamically.
4. Save survey responses in a CSV file for record-keeping, with each row representing a participant's answers.
5. Add an option to reload the application and view cumulative survey results based on previously saved responses.

Discussion and analysis of results:

Conclusion:

Lab Session 12

Objective:

Understanding JavaFX applications: Design and event handling for interactive user interfaces.

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.

Home Task 1: Creating a Role-Based Authentication System

Description: Develop a JavaFX application with a login system that grants role-specific access based on user type (e.g., Admin, Manager, User).

Instructions:

1. Design a login form with the following fields:
   - Username (TextField)
   - Password (PasswordField)
   - Role (ComboBox) with options: Admin, Manager, User
2. Predefine credentials and roles for testing:
   - Admin: admin / admin123
   - Manager: manager / manager123
   - User: user / user123
3. After successful login:
   - Admin should see a dashboard to manage all users.
   - Manager should see a page to view and edit team-specific details.
   - User should see a basic welcome screen.
4. Display error messages for incorrect credentials or role mismatches.

Home Task 2: Scenario: Adding Two-Step Authentication to Login System

Description: Enhance the login system by implementing two-step authentication (2FA) for extra security.

Instructions:

1. After successful username and password verification, prompt the user to enter a one-time password (OTP).
2. Simulate OTP generation:
   - Display the generated OTP in the console for simplicity (e.g., "Your OTP is 1234").
3. Allow users up to 3 attempts to enter the correct OTP.
4. Display a "Login Successful" message for valid OTPs or lock the user out after 3 failed attempts.

Home Task 3: Scenario: Creating a Multi-Step Registration System

Description: Design a multi-step registration system to collect detailed user information in stages.

Instructions:

1. Divide the form into three steps, each displayed on a separate scene:
   - **Step 1**: Basic information (Name, Email, Phone Number).
   - **Step 2**: Address details (Street, City, ZIP Code).
   - **Step 3**: Account details (Username, Password).
2. Add "Next" and "Back" buttons to navigate between steps.
3. Validate input for each step before proceeding to the next.
4. Display a summary of all entered information on the final screen before saving it to a text file.

Discussion and analysis of results:

Conclusion:

Lab Session 13

Objective:

Understanding data storage and retrieval in a signup page using file handling in Java.

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.

Home Task 1: Developing a User Profile Management System

Context: You are creating a platform that stores and manages user profiles, where users can enter and view their personal information, and the platform ensures that the data is properly formatted and displayed.

Task Description:

1. **Create a JavaFX application** with a form that collects the following user details:
   - Full Name, Email Address, Gender, Address, and Date of Birth.
2. **Validate the input** for each field:
   - Ensure that the **email** is in the correct format.
   - Ensure that the **address** field contains both a street address and a city.
   - Ensure that **Date of Birth** is in a valid date format.
3. **Store the user's information** in a text file named user_profiles.txt, with fields separated by commas (e.g., Name, Email, Gender, Address, DOB).
   - Ensure the data is written to the file in a readable format, maintaining consistency across records.
4. **Implement a feature to read the stored user profiles** from user_profiles.txt and display them in a structured format using a JavaFX table.
   - Allow the user to view their profile data and ensure proper formatting in the table (e.g., column headers like "Name", "Email", etc.).
5. Add **error handling** to display a message if the file cannot be read or if any of the fields contain invalid data.

Home Task 2: Developing a Participant Feedback Collection System for an Event

Context: You are creating an application to collect and track feedback from participants at an event. The system should allow users to submit feedback, store it, and provide a summary of the feedback received.

Task Description:

1. **Develop a user interface** to collect participant feedback with the following fields:
   - **Name**, **Email**, and **Feedback Rating** (scale of 1 to 5).
   - A **TextArea** for detailed feedback comments.
2. **Store each record in a file** named feedback_records.txt, with each participant's information saved on a new line in a comma-separated format:
   - Name, Email, Rating, and Feedback Comment.
3. **Implement functionality to read and display the feedback records** from feedback_records.txt:
   - Display all feedback in a structured table format in the application.
   - Show a summary of the total number of feedback submissions, average rating, and any common comments or suggestions.
4. **Add Data Analysis Features**:
   - Display a **pie chart** or **bar chart** representing the distribution of ratings (e.g., percentage of feedbacks for each rating from 1 to 5).
   - Provide a feature to filter feedback by rating, so the user can view feedback that corresponds to a specific rating.
5. **Implement Error Handling**:
   - Handle missing or invalid input by showing an appropriate message.
   - Ensure the system can process incomplete data and provide meaningful feedback or warnings when necessary.

Home Task 3: Designing a Multi-Store Product Inventory System with Supplier Integration

Context: You are building a comprehensive inventory management system for a retail chain with multiple stores. The system should track product availability across stores, manage suppliers, and calculate inventory value based on product quantity and price.

Task Description:

1. **Create a GUI** to input the following product details for each store:
    - **Product Name**, **Quantity**, **Price**, **Store Location**, **Supplier**, and **Description**.
    - Add a **Supplier Contact** field to store the supplier's contact information.
2. **Store product details** in a **CSV file** named multi_store_inventory.csv, with fields separated by commas.
    - Ensure the file stores the following data: Product Name, Quantity, Price, Store Location, Supplier, Supplier Contact, and Description.
3. **Implement functionality** to:
    - Read and display all products across different store locations in a structured table format.
    - Display the following details in the table: Product Name, Quantity, Price, Store Location, and Supplier.
4. **Add a feature to track total inventory value** for each store:
    - Calculate the total value of products in stock for each store by multiplying Quantity and Price.
    - Display the total inventory value for each store in the table.
5. **Implement a search feature** to:
    - Search for products by name or supplier and filter results by store location.
    - Display all matching products across stores, showing the store name and available quantity.
6. **Supplier Management**:
    - Allow users to add new suppliers to the system, updating the Supplier and Supplier Contact fields.
    - Provide a list of suppliers and their contact details.
7. **Generate an Inventory Report**:
    - Allow users to generate a report showing total inventory value by store and by product.
    - Export the report to a **PDF** file or display it in a pop-up dialog.
8. **Error Handling**:
    - Implement robust error handling for missing data, invalid inputs, or duplicate product entries.
    - Display meaningful messages when incorrect or incomplete data is entered.

Discussion and analysis of results:

Conclusion:

Lab Session 14 (Assessment of Open-Ended Lab)

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: ______________