



ALMA-MATER OF QUAID-E-AZAM MOHAMMAD ALI JINNAH
SINDH MADRESSATUL ISLAM UNIVERSITY



SUBMITTED BY:

STUDENT NAME	STUDENT ID
Zuhaib Dayo	BIT – 22F – 019
Syed Qaiser Nawab Sherazi	BIT – 22F – 026
Ali Ahmed Mughal	BIT – 22F – 016

(INFORMATION TECHNOLOGY)

Submitted To: Abdul Kareem Kashif

**SINDH MADRESSA TUL ISLAM UNIVERSITY
KARACHI**

Introduction

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data. Object-oriented programming takes the view that what are really cared about are the objects that is to be manipulated rather than the logic required to manipulate them.

The first step in OOP is to identify all the **objects** the programmer wants to manipulate and how they relate to each other, known as data modeling. Once an object has been identified, it is generalized as a **class** of objects which defines the kind of data it contains and any logic sequences that can manipulate it.

The building of OOP is based on following four pillars.

1. Abstraction
2. Encapsulation.
3. Inheritance.
4. Polymorphism.

Java

Java is a programming language and a platform based on object-oriented programming concepts. It was first released by Sun Microsystems in 1995.

There are lots of applications and websites that will not work unless Java is being installed, and more are created every day. Java is high level, fast, secure, and reliable.

From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Java language will be used throughout this workbook to understand OOP.

Content

Lab No.	Lab Objective
1	Introduction to Java
2	Java Programming Elements
3	Java Control Statement & Operators
4	Introducing Classes and Objects
5	Overloading & Access Control
6	Arrays & Strings
7	Inheritance in Java
8	Use of abstract and final
9	Packages & Interfaces
10	Exception Handling
11	Introducing JavaFx – Java GUI
12	Exploring JavaFx
13	JavaFx Application
14	Data Storage and Retrieval using filing in Java
15	Introducing UML Diagram

Lab Details

Lab No.	Lab Objective	Marks Obtained	Signature
1	Introduction to Java <u>Objective:</u> Getting familiar with the Java development kit (JDK). Running your first Java program using CMD and an IDE.		
Student Feedback:			
2	Java Programming Elements <u>Objective:</u> Learning Input/Output handling on Java console. Understanding variables using primitive and non-primitive data types. Exploring Java's built in classes.		
Student Feedback:			
3	Java Control Statement & Operators <u>Objective:</u> Understanding the control statements of Java including Loops & if-else. Exploring different operators used in Java.		
Student Feedback:			
4	Introducing Classes and Objects <u>Objective:</u> Understanding concepts of class and object in java. Implementing a class with members including data, methods and constructors.		
Student Feedback:			
5	Overloading & Access Control <u>Objective:</u> Understanding concepts method and constructor overloading. Learn how to provide different access controls on class members.		
Student Feedback:			

6	Arrays & Strings <u>Objective:</u> Understanding Arrays, array index, single and multi-dimensional arrays, traversing the array using loop. Getting familiar with the String class of Java.		
Student Feedback:			
7	Inheritance in Java <u>Objective:</u> Understanding the concept of inheritance, the superclass and subclass.		
Student Feedback:			
8	Use of abstract and final <u>Objective:</u> Understand the abstract method & class and final method and class.		
Student Feedback:			
9	Packages & Interfaces <u>Objective:</u> Understanding the concept packages & interfaces of Java.		
Student Feedback:			
10	Exception Handling <u>Objective:</u> Understand how runtime errors are being handled in Java.		
Student Feedback:			
11	Introducing JavaFx– Java GUI <u>Objective:</u> Understand the design principles of graphical user interfaces (GUIs) using layout managers to arrange GUI components. Understand basic component of JavaFx and their interaction used in different program of Java, such as Label, Button, Text Box, Combo Box etc.		
Student Feedback:			

12	Exploring JavaFx <u>Objective:</u> Show different JavaFx Layouts and Charts.		
Student Feedback:			
13	JavaFx Application <u>Objective:</u> Design a Login page using JavaFx components.		
Student Feedback:			
14	Data Storage and Retrieval using Filing in Java <u>Objective:</u> Data storage and retrieval in Signup page using Filing.		
Student Feedback:			
15	Introducing UML diagram <u>Objective:</u> Understand the basics of class diagram of UML.		
Student Feedback:			

Note: Students are advised to write their comments about whether the objective of the lab was achieved in **Student Feedback**.

Introduction to Java

Lab 1

Objective: Getting familiar with the Java development kit (JDK). Running your first Java program using CMD and an IDE.

Theory:

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.

The screenshot shows the Oracle Technology Network website. The main content area is titled "Java SE Development Kit 8 Downloads". It includes a "See also:" section with links to the Java Developer Newsletter, Java Developer Day, and Java Magazine. Below this is a table of available JDK versions and their download links.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.78 MB	jdk-8u111-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.73 MB	jdk-8u111-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.35 MB	jdk-8u111-linux-i586.rpm
Linux x86	175.04 MB	jdk-8u111-linux-i586.tar.gz
Linux x64	158.35 MB	jdk-8u111-linux-x64.rpm
Linux x64	173.04 MB	jdk-8u111-linux-x64.tar.gz
Mac OS X	227.39 MB	jdk-8u111-macosx-x64.dmg

Figure 1.1: Download the JDK from Oracle's website

2. Run the .exe file on your System and follow the steps as given by the installer.



Figure 1.2 Installing JDK

3. Note down the location where j2se has been installed. Java Platform is also called as J2SE (Java 2 Platform Standard Edition)
4. Following screen will appear after successful installation.



Figure 1.3: Successful installation

5. Post installation, you will need to set some environment variables in windows.
6. The path is required to be set for using tools such as javac, java etc. If you are saving the java source file inside the jdk/bin directory, path is not required to be set because all the tools will be available in the current directory. But If you are having your java file outside the jdk/bin folder, it is necessary to set path of JDK.

Setting Java Path

There are 2 ways to set java path:

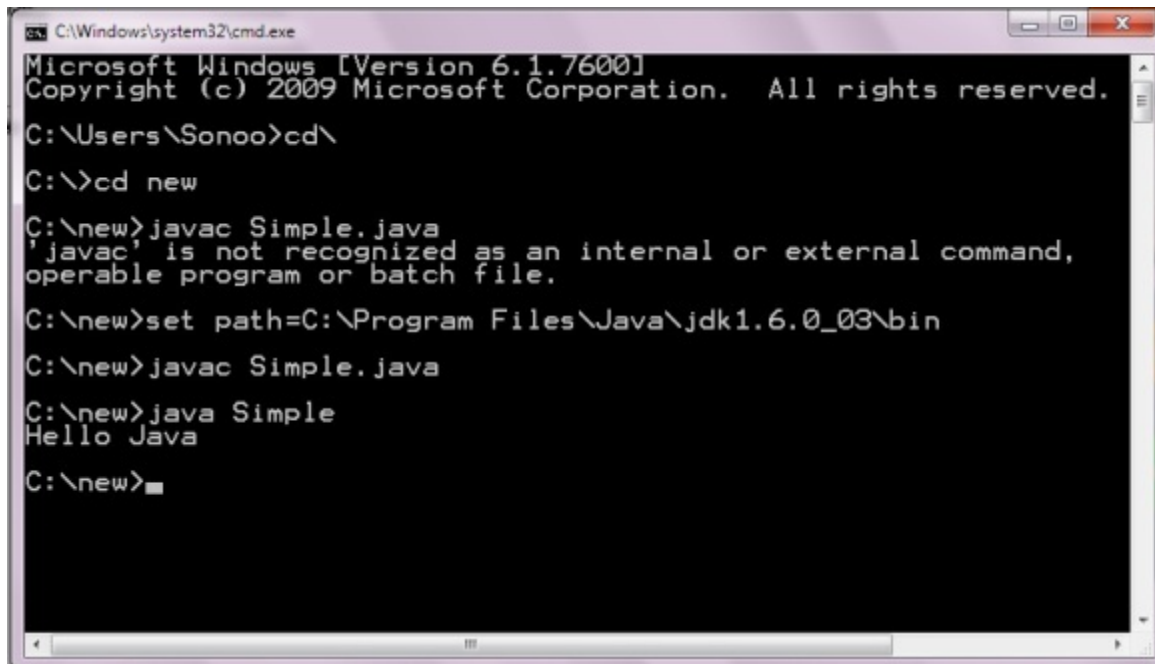
1. temporary
2. permanent

1. How to set Temporary Path of JDK in Windows

To set the temporary path of JDK, you need to follow following steps:

- Open command prompt
- Copy the path of jdk/bin directory
- Write in command prompt: set path=copied_path
For Example:
set path=C:\Program Files\Java\jdk1.6.0_23\bin

Let's see it in the figure given below:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>cd\
C:\>cd new
C:\new>javac Simple.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

C:\new>set path=C:\Program Files\Java\jdk1.6.0_03\bin
C:\new>javac Simple.java
C:\new>java Simple
Hello Java
C:\new>
```

Figure 1.4: Command Prompt

2. How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to
MyComputer properties -> advanced tab -> environment variables -> new tab of user
variable -> write path in variable name -> write path of bin folder in variable value -> ok ->
ok -> ok

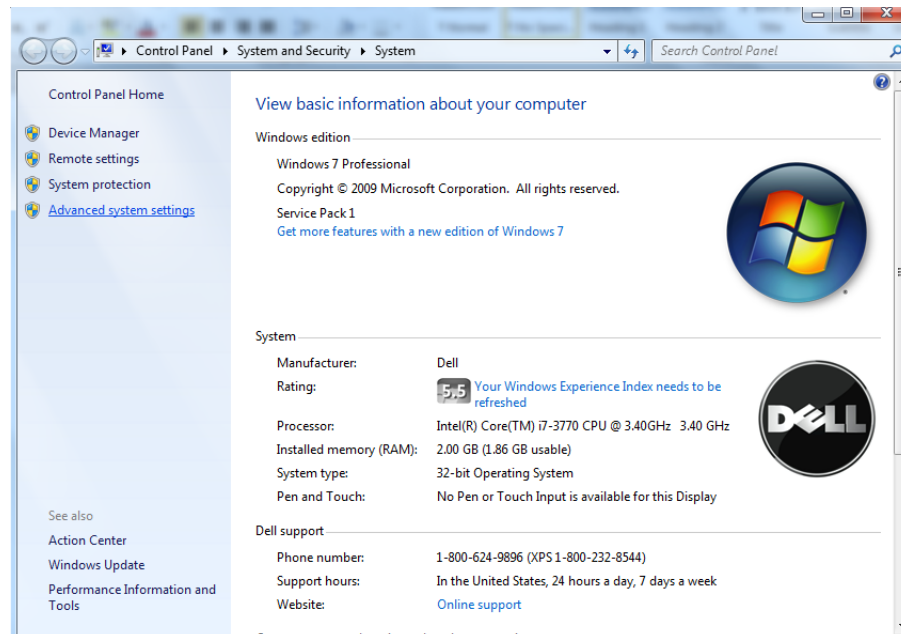


Figure 1.5: Advance System Settings - Windows

- Click on environment variables

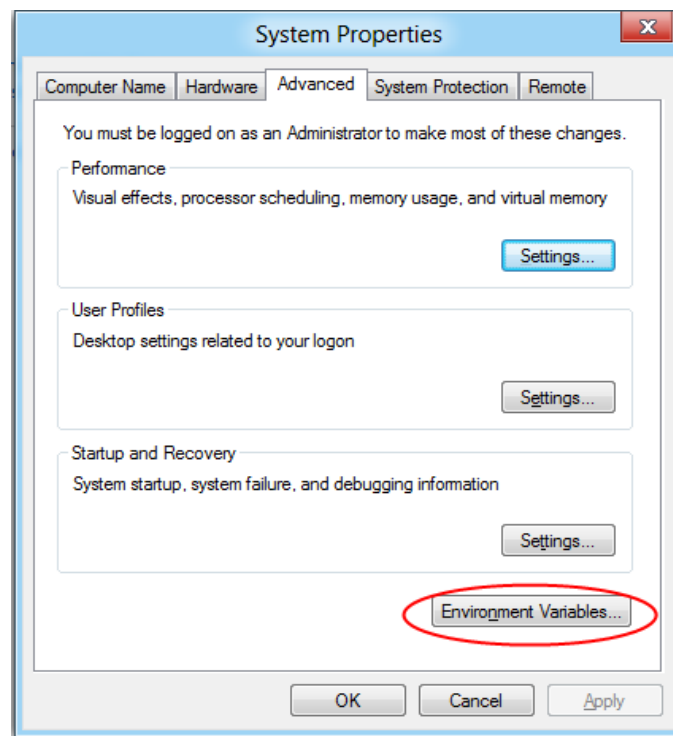


Figure 1.6: Set Environment Variables

- Select Path variable and click on Edit button.

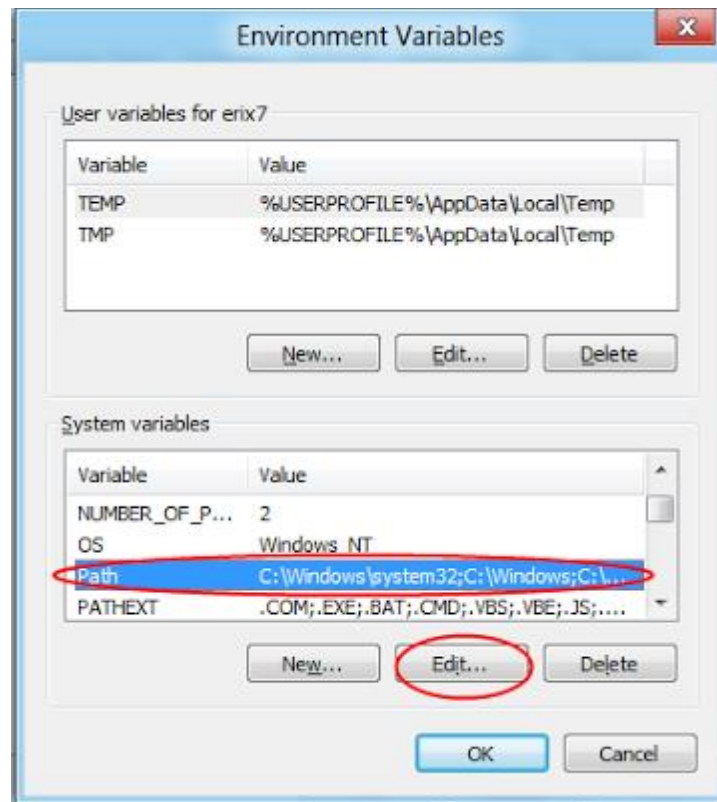


Figure 1.7: Set Path Variable

- Copy the path of the bin folder of the JDK and paste it in the variable value field of the path variable. So add ";C:\Program Files\Java\jdk1.7.0_02\bin" in the path string.

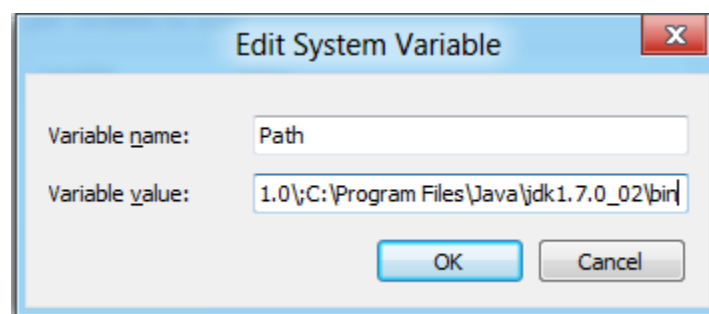


Figure 1.8: Edit System Variable

- Click OK... to finish.
- Now your permanent path is set. You can now execute any program of java from any drive.

Note:

The PATH environment variable is a series of directories separated by semicolons (;) and is not case-sensitive. Microsoft Windows looks for programs in the PATH directories in order, from left to right.

The new path takes effect in each new command window you open after setting the PATH variable.

Lab Task:

Write a simple program in java, compile and run it using cmd.

1. Open notepad and type the content.

```
class HelloWorld {  
    public static void main(String args[]){  
        System.out.println("Hello World!!");  
    }  
}
```

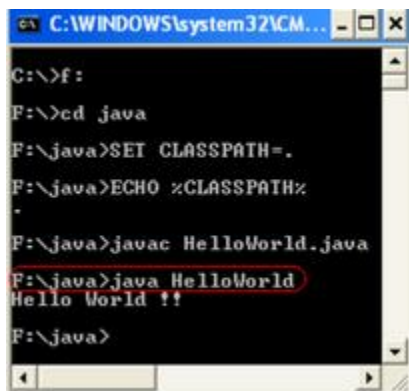
2. Save it as HelloWorld.java. Naming is strictly case-sensitive; make sure you type as it is given above.
3. Now open a command prompt and change to the directory where you have saved HelloWorld.java
4. Enter SET CLASSPATH.
5. Press enter; this sets the classloader path to search for class files in the current directory.
6. Type the command javac to compile your code. If the compilation is successful, you would not be shown any errors.

```
javac HelloWorld.java
```

Type the command java HelloWorld and press enter. Please note that you did not include any extension such as .java or .class to the class name you wish to run.

```
java HelloWorld
```

7. You would get nice output greeting "Hello World!!"



The screenshot shows a Windows Command Prompt window with the title bar 'C:\WINDOWS\system32\cmd.exe'. The command history is as follows:
C:\>f:
F:\>cd java
F:\java>SET CLASSPATH=.
F:\java>ECHO %CLASSPATH%
.
F:\java>javac HelloWorld.java
F:\java>java HelloWorld
Hello World !!
F:\java>

What is an IDE?

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE may consist of a source code editor; build automation tools, compiler and a debugger.

There are different IDE's used for Java development including NetBeans and Eclipse.

Before starting to develop the complex programs, you need an IDE. In this workbook we will be using Eclipse IDE for code writing, compilation and execution.

Install Eclipse IDE

Download Eclipse from its official website and install it on your system by following the instruction provided by the installer.

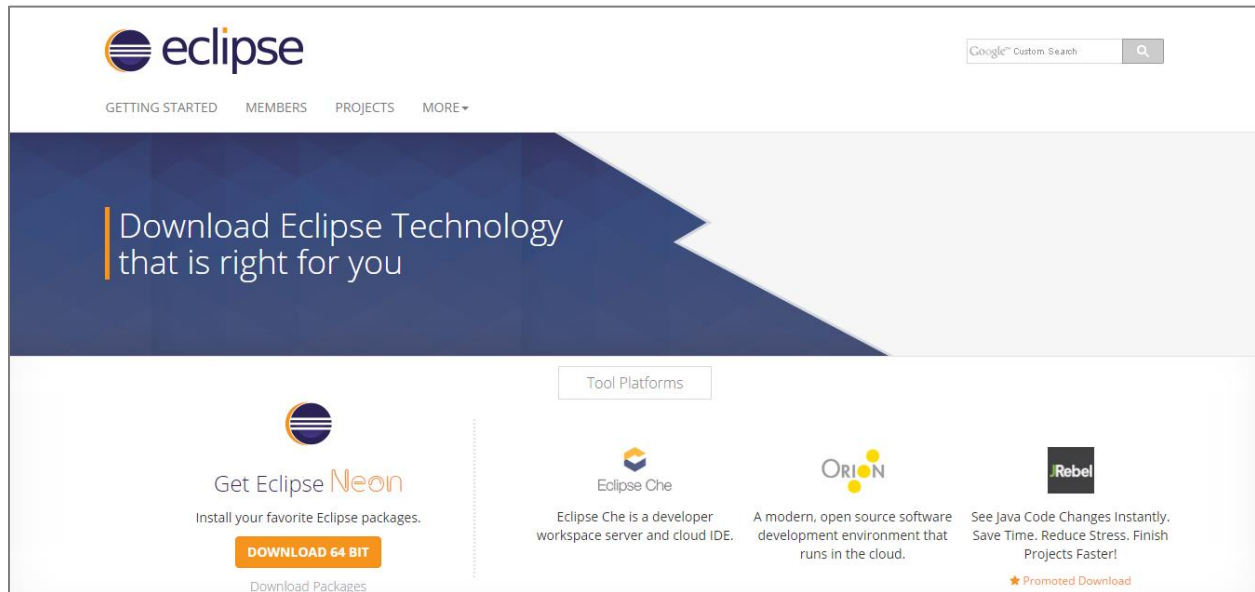


Figure 1.9: Download Eclipse IDE

After installation, open eclipse. You will see the welcome screen.

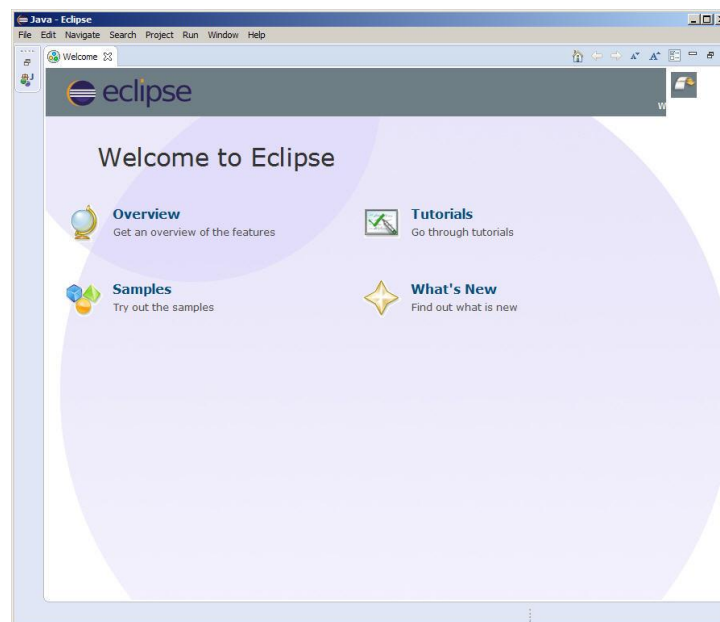


Figure 1.10: Welcome Screen of Eclipse

Write a simple program in java, compile and run it using IDE.

- Create new project in eclipse. Click on file menu, select Project, then select Java Project and click Next. You will see the following screen. Give your project a name and click Finish.

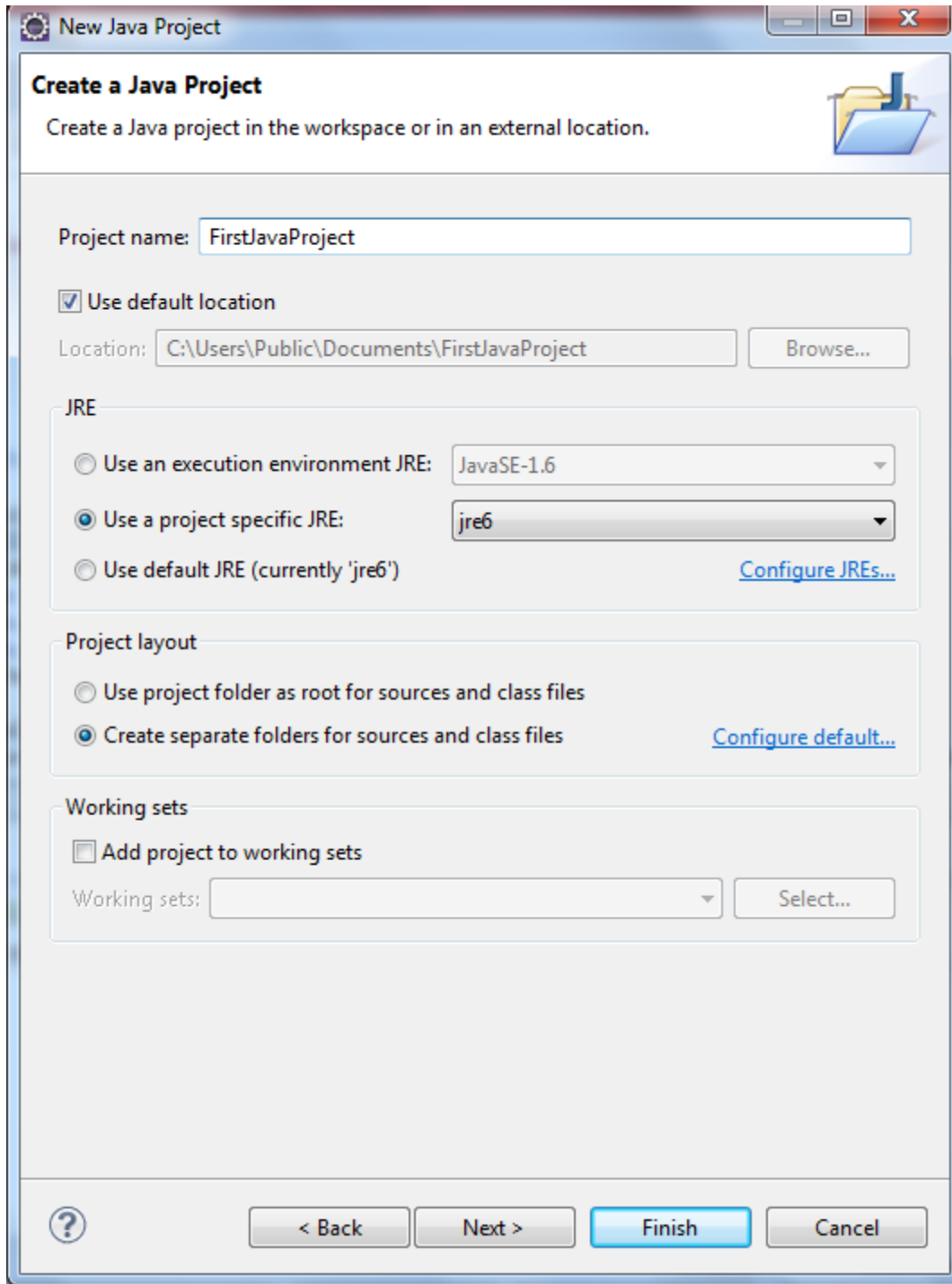


Figure 1.11: Creating new project in Eclipse

- Now right click on your project available in project explorer and create new class.

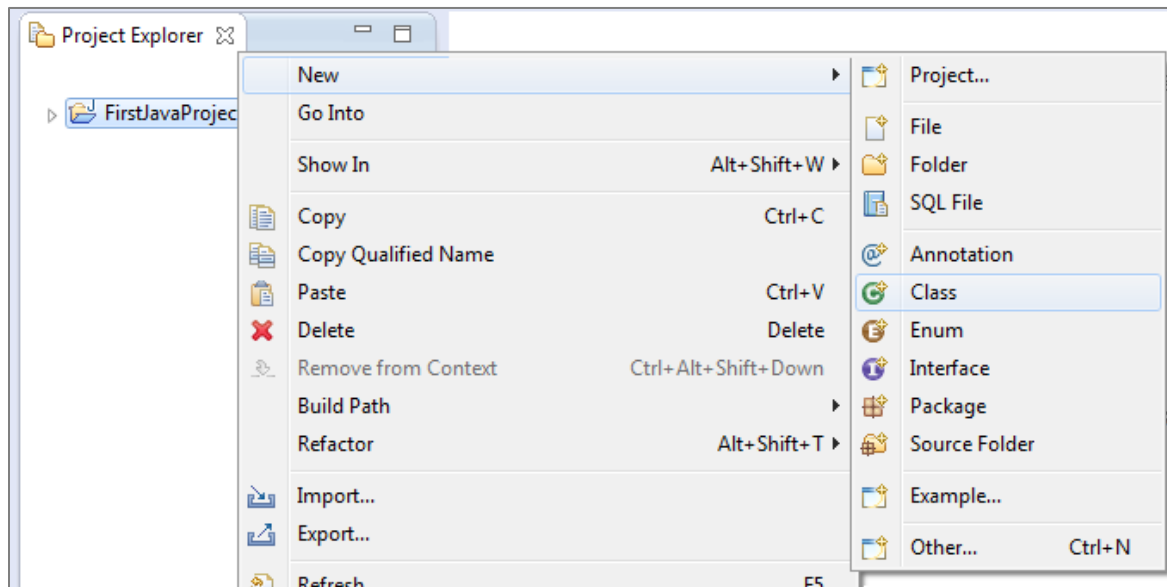


Figure 1.12: Add new Class

- A window similar to one shown below will appear. Enter the class name and check the checkbox for public static void main (String[] args). Click finish.

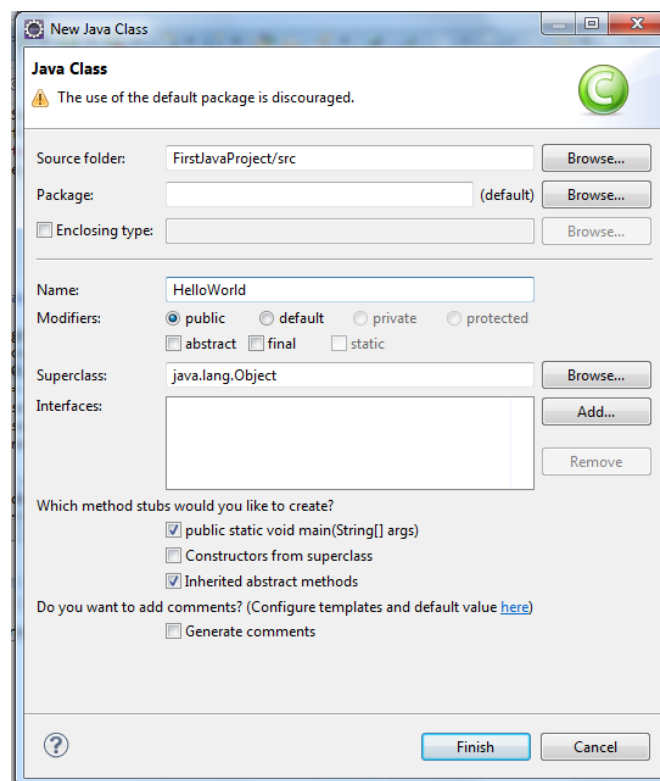


Figure 1.13: Creating new Class

- A java file will open with some added code. Just add the following line inside the main() method.`System.out.println("Hello World!!");`

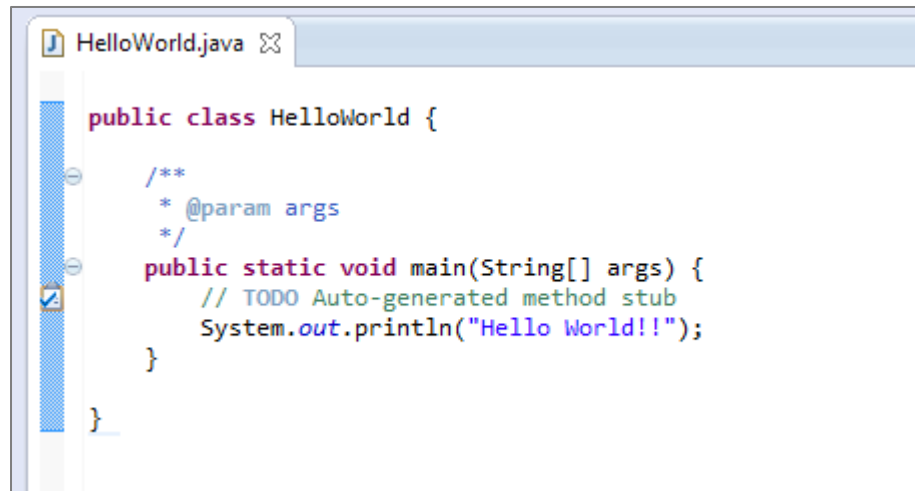


Figure 1.14: Adding Java Code

- Save the file.
- Now compile and execute this code. In Eclipse compilation and Execution performs by clicking the button highlighted in the image below.

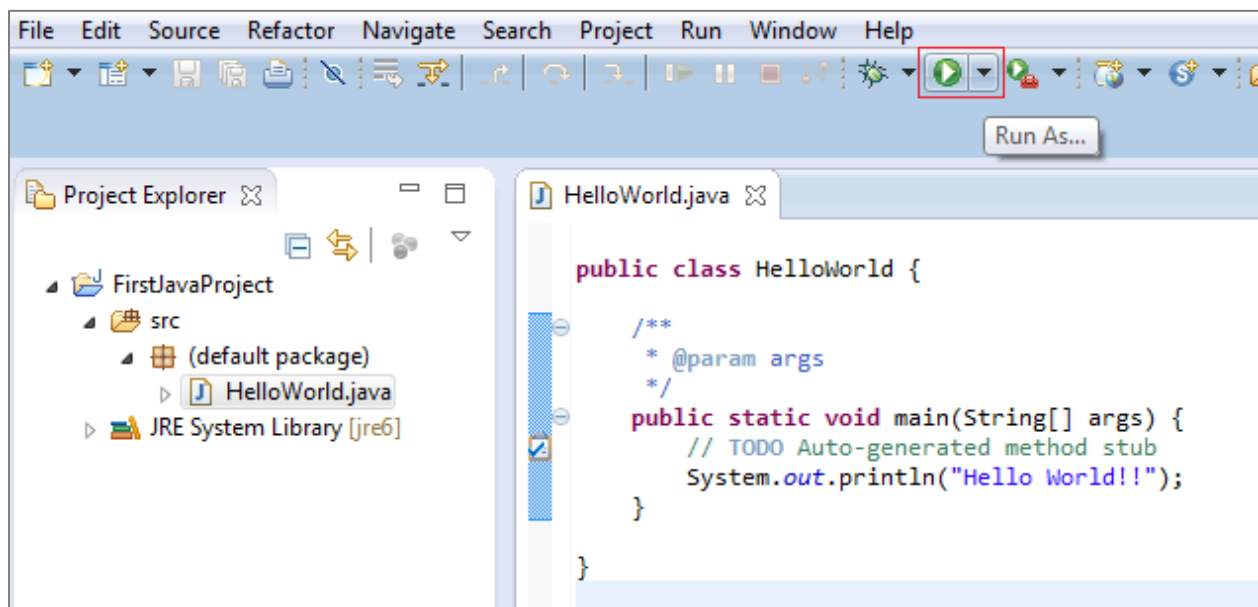


Figure 1.15: compile& Execute

- The output of this code will be generated in the console window present below.

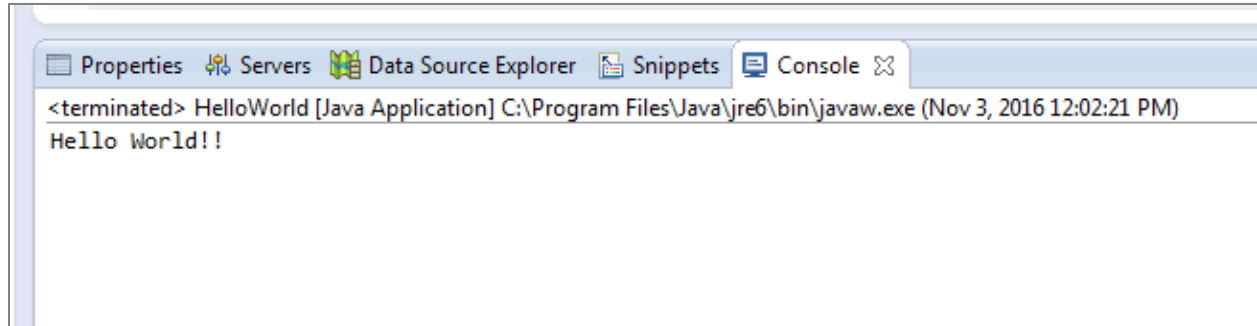


Figure 1.16: Console Output

Lab Assignment:

1. Write a simple java program with command-line argument in java.

```
public class CommandLineArguments {  
    public static void main(String[] args) {  
        // Check if any command-line arguments are provided  
        if (args.length > 0) {  
            System.out.println("Command-line arguments:");  
            // Loop through the arguments and print them  
            for (String arg : args) {  
                System.out.println(arg);  
            }  
        } else {  
            System.out.println("No command-line arguments provided.");  
        }  
    }  
}
```

Java Programming Elements

Lab 2

Objective: Learning Input/Output handling on Java console. Understanding variables using primitive and non-primitive data types. Exploring Java's built in classes.

Theory:

Console input

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

```
Scanner input = new Scanner(System.in);
double radius = input.nextDouble();
```

Import the class by adding

```
import java.util.Scanner;
```

Console output

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

```
System.out.print("Hello ");
System.out.println("world");
```

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

```
System.out.printf("Your Total amount is %4.2f", total);
System.out.printf("count is %d and amount is %f", count, amount);
```

Data Types in Java

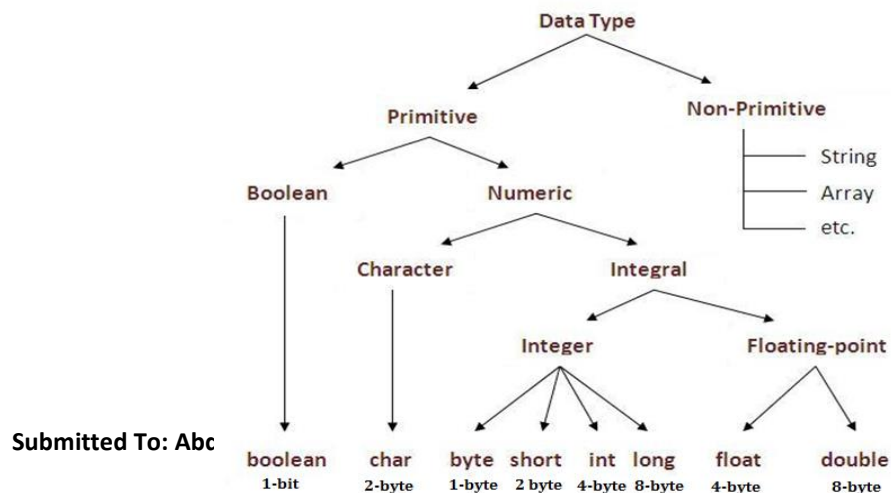
A data type in a programming language is a set of data with values having predefined characteristics. There are two basic types in Java.

1. **Primitive**

A primitive type is predefined by the language and is named by a reserved keyword.

2. **Non-Primitive**

It is a reference data type, which are references to objects



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 2.1: Data Types in Java

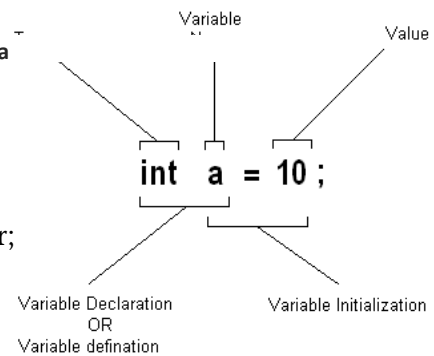


Figure 2.2: Variable Initialization

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.

Trigonometric / 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.

Method	Description
boolean after(Date date)	Returns true if the invoking Date object contains a date that is later than the one specified by <i>date</i> . Otherwise, it returns false .
boolean before(Date date)	Returns true if the invoking Date object contains a date that is earlier than the one specified by <i>date</i> . Otherwise, it returns false .
Object clone()	Duplicates the invoking Date object.
int compareTo(Date date)	Compares the value of the invoking object with that of <i>date</i> . Returns 0 if the values are equal. Returns a negative value if the invoking object is earlier than <i>date</i> . Returns a positive value if the invoking object is later than <i>date</i> .
boolean equals(Object date)	Returns true if the invoking Date object contains the same time and date as the one specified by <i>date</i> . Otherwise, it returns false .
long getTime()	Returns the number of milliseconds that have elapsed since January 1, 1970.
int hashCode()	Returns a hash code for the invoking object.
void setTime(long time)	Sets the time and date as specified by <i>time</i> , which represents an elapsed time in milliseconds from midnight, January 1, 1970.
String toString()	Converts the invoking Date object into a string and returns the result.

Lab Task:

- 1. Write a Java program to take different input from user and store the input in variables with respective data type and then display the data on the console.**

```
import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args)
    {
        String name;
        char gender;
        int age;
        double height;

        Scanner input = new Scanner(System.in);

        System.out.println("Enter Name: ");
        name = input.nextLine();

        System.out.println("Enter Gender (M/F): ");
        gender = input.next().charAt(0);

        System.out.println("Enter Age: ");
        age = input.nextInt();

        System.out.println("Enter Height: ");
        height = input.nextDouble();

        System.out.println(" _____ Student Data _____ \n");

        System.out.println("\t Name: " + name);
        System.out.println("\t Gender: " + gender );
        System.out.println("\t Age: " + age);
        System.out.println("\t Height: " + height);

    }
}
```

- 2. Write a Java program to explore Math class.**

```
public class MathClass
{
    public static void main(String[] args)
    {
        double a=45,b=1,sn,cs,tn,snh,csn,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);

    }
}
```

3. Write a Java program to explore Date class.

```
import java.util.Date;
class DateDemo
{
public static void main(String args[]) {
    // Instantiate a Date object
    Date date = new Date();
    // display time and date using toString()
    System.out.println(date);
    // Display number of milliseconds since midnight, January 1, 1970
    long msec = date.getTime();
    System.out.println("Milliseconds since Jan. 1, 1970 " + msec);
}
}
```

Java Control Statement & Operators

Lab 3

Objective: Understanding the control statements of Java including Loops & if-else. Exploring different operators used in Java.

Lab Assignment:

1. Program the following.

Implement the following equation

$$3x^4\sin(180x) + 4x^3\cos(90x) + x^2\sin(\tan(45)) + 7x + 9\cos(90x^2)$$

Where x may be user defined value.

import java.util.Scanner;

```
public class EquationCalculation {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the value of x
        System.out.print("Enter the value of x: ");
        double x = scanner.nextDouble();

        // Calculate the result using the equation
        double result = (3 * Math.pow(x, 4) * Math.sin(180 * x)) +
            (4 * Math.pow(x, 3) * Math.cos(90 * x)) +
            (Math.pow(x, 2) * Math.sin(Math.tan(45))) +
            (7 * x) +
            (9 * Math.cos(90 * Math.pow(x, 2)));

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

        // Close the Scanner
        scanner.close();
    }
}
```

Theory:

Iteration Statements

A loop can be used to tell a program to execute statements repeatedly. Java provides a powerful construct called a loop that controls how many times an operation or a sequence of operations is performed in succession. A loop can be nested inside another loop. Different form of loops can be nested with one another.

Java provides three types of loop statements:

1. while loops,
2. do-while loops, and
3. for loops.

Loop Syntax

<i>while</i>	<i>do-while</i>	<i>for</i>
<pre>while (loop- continuation- condition) { // Loop body Statement(s); }</pre>	<pre>do { // Loop body; Statement(s); } while (loop-continuation- condition);</pre>	<pre>for (initial-action; loop-continuation- condition; action-after-each- iteration) { // Loop body; Statement(s); }</pre>

Selection Statements

Selection Statements of Java programming language decides the next statement for execution. Selection statements use conditions that are Boolean expressions. A Boolean expression is an expression that evaluates to a Boolean value: true or false. An if statement can be inside another if statement to form a nested if statement.

Java provides three types of selection statements:

1. If statements
2. If-else statements
3. If-else if
4. Switch

if statement executes the statements if the condition is true. if-else statement is two way statement that decides which statements to execute based on whether the condition is true or false. Multi-way **if-else** statement is the preferred coding style for multiple alternative if statements.

If-else Syntax

<i>if</i>	<i>do-while</i>	<i>for</i>
<pre> if (boolean-expression) { statement(s); } </pre>	<pre> if (boolean-expression) { //for-the-true-case; } else { //for-the-false-case; } </pre>	<pre> if (boolean-expression) { // for-1st-case; } else if (boolean-expression) { // for-2nd-case; } else { //for-default-case; } </pre>

Switch Syntax

```

switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
.
.
.
casevalueN:
// statement sequence
break;
default:
// default statement sequence
}

```

Operators in Java

There are many operators provide by Java. Following are the most commonly used operators.

Comparison Operators:

Comparison operators can be used to compare two values. The result of the comparison is a Boolean value: true or false. Java provides six comparison operators.

Comparison Operators		
<i>Java Operator</i>	<i>Mathematics Symbol</i>	<i>Name</i>
<	<	less than
<=	≤	less than or equal to
>	>	greater than
>=	≥	greater than or equal to
==	=	equal to
!=	≠	not equal to

Figure 2.3: Comparison Operators

Logical Operators:

The logical can be used to create a compound Boolean expression. Sometimes, whether a statement is executed is determined by a combination of several conditions.

Boolean Operators		
<i>Operator</i>	<i>Name</i>	<i>Description</i>
!	not	logical negation
&&	and	logical conjunction
	or	logical disjunction
^	exclusive or	logical exclusion

Figure 2.3: Boolean Operators

Lab Task:

1. Write a Java program to use an if-else-if ladder to determine which season a particular month is in.

```
// Demonstrate if-else-if statements.
classIfElse {
public static void main(String args[]) {
    int month = 4; // April
    String season;
    if(month == 12 || month == 1 || month == 2)
        season = "Winter";
    else if(month == 3 || month == 4 || month == 5)
        season = "Spring";
    else if(month == 6 || month == 7 || month == 8)
        season = "Summer";
    else if(month == 9 || month == 10 || month == 11)
        season = "Autumn";
    else
        season = "Invalid Month";
    System.out.println("April is in the " + season + ".");
}
}
```

2. Write a Java program to generate following pattern.

```
// Loops may be nested.
class Nested {
public static void main(String args[]) {
    inti, j;
    for(i=0; i<10; i++) {
        for(j=i; j<10; j++)
            System.out.print(".");
        System.out.println();
    }
}
}
```

Lab Assignment:

1. Write a program 1 of this lab using *switch* statement.

```
import java.util.Scanner;
```

```
public class EquationCalculation {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the value of x
        System.out.print("Enter the value of x: ");
        double x = scanner.nextDouble();

        // Prompt the user to choose the equation option
        System.out.println("Choose an equation option:");
        System.out.println("1. Equation 1");
        System.out.println("2. Equation 2");
        int option = scanner.nextInt();

        // Calculate the result based on the chosen equation option
        double result = 0;

        switch (option) {
            case 1:
                result = 3 * Math.pow(x, 4) * Math.sin(180 * x) +
                    4 * Math.pow(x, 3) * Math.cos(90 * x) +
                    Math.pow(x, 2) * Math.sin(Math.tan(45)) +
                    7 * x +
                    9 * Math.cos(90 * Math.pow(x, 2));
            case 2:
                result = 3 * Math.pow(x, 4) * Math.sin(180 * x) +
                    4 * Math.pow(x, 3) * Math.cos(90 * x) +
                    Math.pow(x, 2) * Math.sin(Math.tan(45)) +
                    7 * x +
                    9 * Math.cos(90 * Math.pow(x, 2));
            default:
                result = 0;
        }
    }
}
```

```
        break;
    case 2:
        result = // Equation 2 implementation
        break;
    default:
        System.out.println("Invalid option.");
        break;
}

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

// Close the Scanner
scanner.close();
}
}
```

2. The Fibonacci sequence is defined by the following rule. The first 2 values in the sequence are 1, 1. Every subsequent value is the sum of the 2 values preceding it. Write a Java Program that print the nth value of the Fibonacci sequence?

```
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the value of n
        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();

        // Calculate and print the nth value of the Fibonacci sequence
        long fibN = calculateFibonacci(n);
        System.out.println("The " + n + "th value of the Fibonacci sequence is: " + fibN);

        // Close the Scanner
        scanner.close();
    }

    public static long calculateFibonacci(int n) {
        if (n <= 0) {
            return 0;
        }
    }
}
```

```
    } else if (n == 1 || n == 2) {  
        return 1;  
    }  
  
    long fibNMinus2 = 1;  
    long fibNMinus1 = 1;  
    long fibN = 0;  
  
    for (int i = 3; i <= n; i++) {  
        fibN = fibNMinus2 + fibNMinus1;  
        fibNMinus2 = fibNMinus1;  
        fibNMinus1 = fibN;  
    }  
  
    return fibN;  
}  
}
```

3. Write a Java program that prompts the user for an integer and then prints out all the prime numbers up to that Integer?

```
import java.util.Scanner;  
  
public class PrimeNumbers {  
    public static void main(String[] args) {  
        // Create a Scanner object to read user input  
        Scanner scanner = new Scanner(System.in);  
  
        // Prompt the user to enter an integer  
        System.out.print("Enter an integer: ");  
        int n = scanner.nextInt();  
  
        System.out.println("Prime numbers up to " + n + ":");  
        // Check each number up to n for primality  
        for (int i = 2; i <= n; i++) {  
            if (isPrime(i)) {  
                System.out.print(i + " ");  
            }  
        }  
  
        // Close the Scanner  
        scanner.close();  
    }  
}
```

```
public static boolean isPrime(int number) {  
    if (number <= 1) {  
        return false;  
    }  
  
    // Check if number is divisible by any number from 2 to its square root  
    for (int i = 2; i <= Math.sqrt(number); i++) {  
        if (number % i == 0) {  
            return false;  
        }  
    }  
  
    return true;  
}
```

Classes & Objects

Lab 4

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

Theory:

Class

A class consists of

- Data(variables)
- Methods
- Constructors

Lab Task:

```
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();
Box mybox2 = new Box();
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 is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Adding Constructor

```
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;
    }
}
```

Lab Assignment:

1. Create a class Calculator with following details;

- The class should contain default and parameterized constructor. The constructors should just print the statement as follows.
 - “Inside Default Constructor”
 - “Inside Parameterized Constructor”.
- The class should contain following data fields and methods;
 - int square = 2 ;
 - int cube = 3;
 - calculateSquare(int x)
 - calculateCube(int x)
 - calculateFactorial (int x)
 - generateTable(int x)

```
public class Calculator {
    private int square;
    private int cube;

    // Default constructor
    public Calculator() {
        System.out.println("Inside Default Constructor");
    }

    // Parameterized constructor
    public Calculator(int square, int cube) {
        this.square = square;
        this.cube = cube;
    }
}
```

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

    public void calculateSquare(int x) {
        int result = x * x;
        System.out.println("Square of " + x + " is: " + result);
    }

    public void calculateCube(int x) {
        int result = x * x * x;
        System.out.println("Cube of " + x + " is: " + result);
    }

    public void calculateFactorial(int x) {
        int result = 1;
        for (int i = 1; i <= x; i++) {
            result *= i;
        }
        System.out.println("Factorial of " + x + " is: " + result);
    }

    public void generateTable(int x) {
        System.out.println("Table of " + x + ":");
        for (int i = 1; i <= 10; i++) {
            System.out.println(x + " * " + i + " = " + (x * i));
        }
    }
}
```

- Create objects of this class using both Constructors in main Class.
- Call all three functions via object.

```
public class Main {
    public static void main(String[] args) {
        // Create an object using the default constructor
        Calculator calculator1 = new Calculator();

        // Call methods on the object
        calculator1.calculateSquare(5);
        calculator1.calculateCube(4);
        calculator1.calculateFactorial(6);
        calculator1.generateTable(3);

        System.out.println();

        // Create an object using the parameterized constructor
        Calculator calculator2 = new Calculator(2, 3);

        // Call methods on the object
        calculator2.calculateSquare(8);
    }
}
```



```
calculator2.calculateCube(10);  
calculator2.calculateFactorial(4);  
calculator2.generateTable(7);  
}  
}
```

Overloading & Access Control

Lab 5

Objective: Understanding concepts method and constructor overloading. Learn how to provide different access controls on class members

Theory:

Method Overloading

If a class has multiple methods by same name but different parameters, it is known as Method Overloading.

Constructor Overloading

If a class has multiple constructors having different parameters, it is known as Constructor Overloading.

Access Control

The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class. There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

Lab Task:

```
// Demonstrate method overloading.
class OverloadDemo {
    void test() {
        System.out.println("No parameters");
    }
    // Overload test for one integer parameter.
    void test(int a) {
        System.out.println("a: " + a);
    }
    // Overload test for two integer parameters.
    void test(int a, int b) {
        System.out.println("a and b: " + a + " " + b);
    }
    // overload test for a double parameter
    double test(double a) {
        System.out.println("double a: " + a);
        return a*a;
    }
}
OverloadDemo() {
    System.out.println("No-args constructor ");
}
OverloadDemo(int demo) {
```

```
        System.out.println("Parameterized Constructor  :" + demo) ;
    }
}
// _____ Calling Class _____

class Overload {
public static void main(String args[])
{
    OverloadDemo ob = new OverloadDemo();
    OverloadDemo ob1 = new OverloadDemo(33);

    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);
}
}
```

Lab Assignment:

1. Design a class named Account that contains:

- A private int data field named id for the account (default 0).
- A private double data field named balance for the account (default 0).
- A private double data field named annualInterestRate that stores the current interest rate (default 0). Assume all accounts have the same interest rate.
- A private Date data field named dateCreated that stores the date when the account was created.
- A no-arg constructor that creates a default account.
- A constructor that creates an account with the specified id and initial balance.
- The accessor and mutator methods for id, balance, and annualInterestRate.
- The accessor method for dateCreated.
- A method named getMonthlyInterestRate() that returns the monthly interest rate.
- A method named getMonthlyInterest() that returns the monthly interest.
- A method named withdraws that withdraws a specified amount from the account.
- A method named deposit that deposits a specified amount to the account.

(Hint: Monthly interest is balance * monthlyInterestRate.

monthlyInterestRate is annualInterestRate / 12.

Note that annualInterestRate is a percentage. You need to divide it by 100.

```
import java.util.Date;
```

```
public class Account {
    private int id;
    private double balance;
    private double annualInterestRate;
```

```
private Date dateCreated;

public Account() {
    id = 0;
    balance = 0;
    annualInterestRate = 0;
    dateCreated = new Date();
}

public Account(int id, double balance) {
    this.id = id;
    this.balance = balance;
    annualInterestRate = 0;
    dateCreated = new Date();
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public double getBalance() {
    return balance;
}

public void setBalance(double balance) {
    this.balance = balance;
}

public double getAnnualInterestRate() {
    return annualInterestRate;
}

public void setAnnualInterestRate(double annualInterestRate) {
    this.annualInterestRate = annualInterestRate;
}

public Date getDateCreated() {
    return dateCreated;
}

public double getMonthlyInterestRate() {
    return annualInterestRate / 12;
}

public double getMonthlyInterest() {
    double monthlyInterestRate = getMonthlyInterestRate();
    return balance * (monthlyInterestRate / 100);
}
```

```
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance.");
        }
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        } else {
            System.out.println("Invalid amount.");
        }
    }
}
```

Write a test program that creates an Account object with an account ID of 1122, a balance of \$20,000, and an annual interest rate of 4.5%. Use the withdraw method to withdraw \$2,500, use the deposit method to deposit \$3,000, and print the balance, the monthly interest, and the date when this account was created.

```
import java.text.DecimalFormat;
```

```
public class TestAccount {

    public static void main(String[] args) {

        // Create an Account object

        Account account = new Account(1122, 20000);

        account.setAnnualInterestRate(4.5);

        // Withdraw $2,500

        account.withdraw(2500);

        // Deposit $3,000

        account.deposit(3000);
```

```
// Print the account details

DecimalFormat decimalFormat = new DecimalFormat("#,##0.00");

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

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

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

}

}
```

Arrays & Strings

Lab 6

Objective: Understanding Arrays, array index, single and multi-dimensional arrays, traversing the array using loop. Getting familiar with the String class of Java.

Theory:

Array

A single array variable can reference a large collection of data. Java and most other high-level languages provide a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. Once an array is created, its size is fixed. An array reference variable is used to access the elements in an array using an index

Syntax:

```
elementType[] arrayRefVar = new elementType[arraySize]; // 1d array
elementType[][] arrayRefVar;                             // 2d array
```

To assign values to the elements, use the syntax:

```
arrayRefVar[index] = value;           // 1d array
arrayRefVar[row][column] = value;     // 2d array
```

Strings

The classes String, StringBuilder, and StringBuffer are used for processing strings. A string is a sequence of characters. Strings are frequently used in programming. In many languages, strings are treated as an array of characters, but in Java a string is treated as an object. A String object is immutable; its contents cannot be changed.

Syntax:

```
String newString = new String(stringLiteral);
String newString = stringLiteral;
```

The String class provides the methods for comparing strings.

Methods	Description
equals(StringLiteral)	Returns true if this string is equal to string s1.
equalsIgnoreCase(StringLiteral)	Returns true if this string is equal to string s1 case insensitive.
compareTo(StringLiteral)	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1

Lab Tasks:

- 1. Write a program using one-dimensional array of numbers and finds the average of a set of numbers.**

```
// Average an array of values.
class Average {
public static void main(String args[]) {
    double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
    double result = 0;
    int i;
    for(i=0; i<5; i++)
        result = result + nums[i];
    System.out.println("Average is " + result / nums.length);
}
}
```

- 2. Write a program to initialize the elements of 2D array with random numbers.**

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

for (int row = 0; row < array2d.length; row++) {
    for (int col = 0; col < array2d.length; col++) {
        array2d[row][col] = Math.round((Math.random()) * 100));
    }
}

for (int row = 0; row < array2d.length; row++) {
    for (int col = 0; col < array2d.length; col++) {
        System.out.print(array2d[row][col] + " ");
    }
    System.out.print("\n");
}
```

- 3. Write a program to explore different methods of String class.**

```
class getCharsDemo {
public static void main(String args[])
{
    String longStr = "This could have been " +
                    "a very long line that would have " +
                    "wrapped around. But string concatenation " +
                    "prevents this.";
    System.out.println(longStr);

    String s = "This is a demo of the getChars method.";
    int start = 10;
    int 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 = "Good-bye";
        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 + " equals " + s4 + " -> " +
            s1.equals(s4));
        System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
            s1.equalsIgnoreCase(s4));
    }
}
```

4. Write a program to sort an array of strings.

```
// A bubble sort for Strings.
class SortString {
    static String arr[] = {
        "Now", "is", "the", "time", "for", "all", "good", "men",
        "to", "come", "to", "the", "aid", "of", "their", "country"
    };

    public static void main(String args[]) {
        for(int j = 0; j < arr.length; j++) {
            for(int i = j + 1; i < arr.length; i++) {
                if(arr[i].compareTo(arr[j]) < 0) {
                    String t = arr[j];

                    arr[j] = arr[i];
                    arr[i] = t;
                }
            }
            System.out.println(arr[j]);
        }
    }
}
```

Lab Assignment:

1. Write a Java program that checks whether a given string is a palindrome or not. Ex: MADAM is a palindrome?

```
import java.util.Scanner;
```

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

```
// Create a Scanner object to read user input
Scanner scanner = new Scanner(System.in);

// Prompt the user to enter a string
System.out.print("Enter a string: ");
String input = scanner.nextLine();

// Check if the string is a palindrome
if (isPalindrome(input)) {
    System.out.println("The string \"" + input + "\" is a palindrome.");
} else {
    System.out.println("The string \"" + input + "\" is not a palindrome.");
}

// Close the Scanner
scanner.close();
}

public static boolean isPalindrome(String str) {
    // Remove any non-alphanumeric characters from the string and convert it to lowercase
    String processedStr = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();

    // Check if the processed string is a palindrome
    int left = 0;
    int right = processedStr.length() - 1;
    while (left < right) {
        if (processedStr.charAt(left) != processedStr.charAt(right)) {
            return false;
        }
        left++;
        right--;
    }

    return true;
}
}
```

2. Write a Java program for sorting a given array of numbers in ascending order?

```
import java.util.Arrays;

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

```
// Define an array of numbers
int[] numbers = { 5, 2, 8, 1, 3 };

// Print the original array
System.out.println("Original Array: " + Arrays.toString(numbers));

// Sort the array in ascending order
bubbleSort(numbers);

// Print the sorted array
System.out.println("Sorted Array: " + Arrays.toString(numbers));
}

public static void bubbleSort(int[] arr) {
    int n = arr.length;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
}
```

3. Write a Java program to multiply two given matrices using 2D array.

```
public class MatrixMultiplier {
    public static void main(String[] args) {
        // Define the matrices
        int[][] matrix1 = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        int[][] matrix2 = {
            {9, 8, 7},
            {6, 5, 4},
        };
    }
}
```

```
        {3, 2, 1}
    };

    // Perform matrix multiplication
    int[][] result = multiplyMatrices(matrix1, matrix2);

    // Display the result
    System.out.println("Matrix Multiplication Result:");
    printMatrix(result);
}

public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) {
    int rows1 = matrix1.length;
    int cols1 = matrix1[0].length;
    int cols2 = matrix2[0].length;

    int[][] result = new int[rows1][cols2];

    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
            int sum = 0;
            for (int k = 0; k < cols1; k++) {
                sum += matrix1[i][k] * matrix2[k][j];
            }
            result[i][j] = sum;
        }
    }

    return result;
}

public static void printMatrix(int[][] matrix) {
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println();
    }
}
```

Inheritance in Java

Lab 7

Objective: Understanding the concept of inheritance, the superclass and subclass.

Theory:

Inheritance

Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a **superclass**. The class that does the inheriting is called a **subclass**. Therefore, a subclass is a specialized version of a superclass. It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements.

Multilevel Inheritance

You can build hierarchies that contain as many layers of inheritance as you like. As mentioned, it is perfectly acceptable to use a subclass as a superclass of another. For example, three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. When this type of situation occurs, each subclass inherits all of the traits found in all of its superclasses. In this case, C inherits all aspects of B and A.

Method Overriding

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden.

There are situations when a superclass is created that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement.

Lab Task:

```
// This program uses inheritance to extend Box.
class Box {
    double width;
    double height;
    double depth;
    // construct clone of an object
    Box(Box ob) { // pass object to constructor
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }
    // constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
```

```
depth = d;
}
// constructor used when no dimensions specified
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
// constructor used when 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);
}
}
```

Multilevel Inheritance

```
// Add shipping costs.
class Shipment extends BoxWeight {
double cost;
// construct clone of an object
Shipment(Shipment ob) { // pass object to constructor
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 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);
}
}
```

Lab Assignment:

1. Design a class named Person and its two subclasses named Student and Employee. Make Faculty and Staff subclasses of Employee.

(The Person, Student, Employee, Faculty, and Staff classes)

A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date hired. A faculty member has office hours and a rank. A staff member has a title. Override the toString method in each class to display the class name and the person's name.

Write a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their toString() methods.

```
class Person {
    private String name;
    private String address;
    private String phoneNumber;
    private String emailAddress;

    public Person(String name, String address, String phoneNumber, String emailAddress) {
        this.name = name;
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.emailAddress = emailAddress;
    }

    @Override
    public String toString() {
        return "Person - Name: " + name;
    }
}

class Student extends Person {
    private static final String CLASS_STATUS = "Freshman";

    public Student(String name, String address, String phoneNumber, String emailAddress) {
        super(name, address, phoneNumber, emailAddress);
    }

    @Override
    public String toString() {
        return "Student - Name: " + super.toString();
    }
}

class Employee extends Person {
```



```
private String office;
private double salary;
private String dateHired;

public Employee(String name, String address, String phoneNumber, String emailAddress,
String office, double salary, String dateHired) {
    super(name, address, phoneNumber, emailAddress);
    this.office = office;
    this.salary = salary;
    this.dateHired = dateHired;
}

@Override
public String toString() {
    return "Employee - Name: " + super.toString();
}
}

class Faculty extends Employee {
    private String officeHours;
    private String rank;

    public Faculty(String name, String address, String phoneNumber, String emailAddress,
String office, double salary, String dateHired, String officeHours, String rank) {
        super(name, address, phoneNumber, emailAddress, office, salary, dateHired);
        this.officeHours = officeHours;
        this.rank = rank;
    }

    @Override
    public String toString() {
        return "Faculty - Name: " + super.toString();
    }
}

class Staff extends Employee {
    private String title;

    public Staff(String name, String address, String phoneNumber, String emailAddress,
String office, double salary, String dateHired, String title) {
        super(name, address, phoneNumber, emailAddress, office, salary, dateHired);
        this.title = title;
    }

    @Override
    public String toString() {
        return "Staff - Name: " + super.toString();
    }
}
```

```
public class TestPerson {  
    public static void main(String[] args) {  
        Person person = new Person("John Doe", "123 Main St", "123-456-7890",  
"john.doe@example.com");  
        System.out.println(person);  
  
        Student student = new Student("Jane Smith", "456 Elm St", "987-654-3210",  
"jane.smith@example.com");  
        System.out.println(student);  
  
        Employee employee = new Employee("Mike Johnson", "789 Oak St", "456-789-0123",  
"mike.johnson@example.com", "Office A", 50000.0, "2021-01-01");  
        System.out.println(employee);  
  
        Faculty faculty = new Faculty("Sarah Davis", "987 Pine St", "321-654-0987",  
"sarah.davis@example.com", "Office B", 60000.0, "2020-05-01", "9 AM - 5 PM",  
"Associate Professor");  
        System.out.println(faculty);  
  
        Staff staff = new Staff("Robert Wilson", "654 Maple St", "789-012-3456",  
"robert.wilson@example.com", "Office C", 40000.0, "2022-02-01", "Manager");  
        System.out.println(staff);  
    }  
}
```

Use of abstract & final

Lab 8

Objective: Understand the abstract method & class and final method and class.

Theory:

Abstract Methods

There are cases when we have to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

In order to ensure that a subclass does, indeed, override all necessary methods, Java provides abstract methods. Certain methods can be made mandatory to be overridden by subclasses by specifying the abstract type modifier.

To declare an abstract method, use this general form:

```
abstract type name(parameter-list);
```

Abstract Class

Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, ***abstract*** keyword is used in front of the class keyword at the beginning of the class declaration.

```
abstract class class_name {  
    abstract type method();  
}
```

There can be **no** objects of an abstract class.

Final Method

final can be use to prevent overriding. To disallow a method from being overridden, specify ***final*** as a modifier at the start of its declaration. Methods declared as final cannot be overridden.

To declare final method, use this general form:

```
final type name(parameter-list){}
```

Final Class

final can be use to prevent inheritance. In order to prevent a class from being inherited, the class is made final. Declaring a class as final implicitly declares all of its methods as final, too.

To do this, precede the class declaration with ***final***.

```
final class class_name { }
```

It is illegal to declare a class as both abstract and final.

Lab Task:

```
// A Simple demonstration of abstract.
abstract class A {
    abstract void callme();
    // concrete methods are still allowed in abstract classes
    void callmetoo() {
        System.out.println("This is a concrete method.");
    }
}

class B extends A {
    void callme() {
        System.out.println("B's implementation of callme.");
    }
}

class AbstractDemo {
    public static void main(String args[]) {
        B b = new B();
        b.callme();
        b.callmetoo();
    }
}
```

Using *final* to Prevent Overriding

```
class A {
    final void meth() {
        System.out.println("This is a final method.");
    }
}

class B extends A {
    void meth() { // ERROR! Can't override.
        System.out.println("Illegal!");
    }
}
```

Using *final* to Prevent Inheritance

```
final class A {
    void meth() {
        System.out.println("This is by default final method.");
    }
}

class B extends A {
    void meth() { // ERROR! Can't override.
        System.out.println("Illegal!");
    }
}
```

Lab Assignment:

1. Consider developing a simple bank application as per given details.

The project contains different classes. One class is **Account** which is abstract. The Account class should implement following details;

- List of attributes:
protected String id;
protected double balance;
- Implement the following constructor:
public Account(String id, double balance) // this constructor will be used from a sub-class's constructor.
- List of methods:
public String getID() // Returns the account id.
public double getBalance() // Returns the current balance.
public *abstract* boolean withdraw(double amount)
public *abstract* void deposit(double amount)

Second class is **SavingsAccount** which extends from Account. The SavingsAccount class should implement following details;

- Implement the following constructor:
public SavingsAccount(String id, double initialDeposit): // the initial deposit passed will be at least \$10.
- List of methods:
public void deposit(double amount): // The provided amount is added to the account
public boolean withdraw(double amount):
Implement the withdraw method to take out the provided amount from the account balance. Incorporate the transaction fee \$2 per withdrawal. A withdrawal that potentially lowers the balance below \$10 is not allowed. The balance remains unchanged but the method returns false. If the withdrawal succeeds, the method returns true.

```
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);
}

class SavingsAccount extends Account {
    private static final double MIN_BALANCE = 10.0;
    private static final double TRANSACTION_FEE = 2.0;

    public SavingsAccount(String id, double initialDeposit) {
        super(id, initialDeposit);
    }

    @Override
    public void deposit(double amount) {
        balance += amount;
    }

    @Override
    public boolean withdraw(double amount) {
        if (balance - amount - TRANSACTION_FEE >= MIN_BALANCE) {
            balance -= amount + TRANSACTION_FEE;
            return true;
        } else {
            return false;
        }
    }
}
```

Packages & Interfaces

Lab 9

Objective: Understanding the concept packages & interfaces of Java.

Theory:

Packages

Packages are containers for classes that are used to keep the class name space compartmentalized. For example, a package allows you to create a class named List, which you can store in your own package without concern that it will collide with some other class named List stored elsewhere. Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.

This is the general form of the package statement:

```
package pkg;
```

This is the general form of the import statement:

```
import pkg1[.pkg2].(classname|*);
```

Interface

Using interface, you can specify a set of methods that can be implemented by one or more classes. The interface, itself, does not actually define any implementation. Although they are similar to abstract classes, interfaces have an additional capability: A class can implement more than one interface. By contrast, a class can only inherit a single superclass (abstract or otherwise).

Lab Task:

```
// A simple package
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();
}
}
```

Call this file AccountBalance.java and put it in a directory called MyPack. Next, compile the file. Make sure that the resulting .class file is also in the MyPack directory. Then, try executing the AccountBalance class, using the following command line:

```
javaMyPack.AccountBalance
```

Remember, you will need to be in the directory above MyPack when you execute this command. (Alternatively, you can use one of the other two options described in the preceding section to specify the path MyPack.)

As explained, AccountBalance is now part of the package MyPack. This means that it cannot be executed by itself. That is, you cannot use this command line:

```
javaAccountBalance
```

AccountBalance must be qualified with its package name.

Interface implementation

```
interface Callback {
    void callback(int param);
}

class Client implements Callback {
    // Implement Callback's interface
    public void callback(int p) {
        System.out.println("callback called with " + p);
    }
}
```

Lab Assignment:

1. Write an interface Crawlable with method crawl. Create another interface Moveable with method move. Then write a class Animal and implement both interfaces to show multiple inheritance.

```
interface Crawlable {
    void crawl();
}
```

```
interface Moveable {
    void move();
}
```

```
class Animal implements Crawlable, Moveable {
    @Override
    public void crawl() {
```



```
        System.out.println("Animal is crawling.");
    }

    @Override
    public void move() {
        System.out.println("Animal is moving.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.crawl();
        animal.move();
    }
}
```

2. Write a class that implements the CharSequence interface found in the java.lang package. Your implementation should return the string backwards. Select one of the sentences from this book to use as the data. Write a small main method to test your class; make sure to call all four methods.

```
public class BackwardsCharSequence implements CharSequence {
    private String data;

    public BackwardsCharSequence(String data) {
        this.data = data;
    }

    @Override
    public int length() {
        return data.length();
    }

    @Override
    public char charAt(int index) {
        return data.charAt(data.length() - 1 - index);
    }

    @Override
    public CharSequence subSequence(int start, int end) {
        StringBuilder sb = new StringBuilder(data.subSequence(start, end));
        return sb.reverse().toString();
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder(data);
        return sb.reverse().toString();
    }
}
```

```
}

public static void main(String[] args) {
    String sentence = "The quick brown fox jumps over the lazy dog.";

    BackwardsCharSequence sequence = new BackwardsCharSequence(sentence);

    System.out.println("Original Sentence: " + sentence);
    System.out.println("Length: " + sequence.length());
    System.out.println("Char at index 10: " + sequence.charAt(10));
    System.out.println("Subsequence from index 5 to 15: " + sequence.subSequence(5, 15));
    System.out.println("Backwards Sequence: " + sequence.toString());
}
}
```

LAB 11

Theory:

An exception is an abnormal condition that arises in a code sequence at run time. In other words, an exception is a run-time error. In computer languages that do not support exception handling, errors must be checked and handled manually—typically through the use of error codes, and so on. This approach is as cumbersome as it is troublesome. Java's exception handling avoids these problems and, in the process, brings run-time error management into the object oriented world.

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed.

Java exception handling is managed via five keywords: *try*, *catch*, *throw*, *throws*, and *finally*.

Lab Task:

To guard against and handle a run-time error, simply enclose the code that you want to monitor inside a try block. Immediately following the try block, includes a catch clause that specifies the exception type

that you wish to catch. To illustrate how easily this can be done, the following program includes a try block and a catch clause that processes the ArithmeticException generated by the division-by-zero error:

```
class ExceptionHandling {
public static void main(String args[]) {
int d, a;
try
{ // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
}
catch (ArithmeticException e)
{ // catch divide-by-zero error
System.out.println("Division by zero. " + e);
}
finally {
System.out.println("procB's finally");
}
System.out.println("After catch statement.");
}
}
```

Here is a sample program that creates and throws an exception. The handler that catches the exception rethrows it to the outer handler.

```
// Demonstrate throw.
class ThrowDemo {
static void demoproc() {
try {
throw new NullPointerException("demo");
} catch(NullPointerException e) {
System.out.println("Caught inside demoproc.");
throw e; // rethrow the exception
}
}
public static void main(String args[]) {
try {
demoproc();
} catch(NullPointerException e) {
System.out.println("Recought: " + e);
}
}
}
```

Lab Assignment:

1. **Write a program to implement calculator's basic functionality with an exception handler that deals with nonnumeric operands; then write another program without using an exception handler to achieve the same objective. Your program should display a message that informs the user of the wrong operand type before exiting.**

```
import java.util.Scanner;

public class CalculatorWithExceptionHandler {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Enter the first number: ");
            double num1 = Double.parseDouble(scanner.nextLine());

            System.out.print("Enter the operator (+, -, *, /): ");
            String operator = scanner.nextLine();

            System.out.print("Enter the second number: ");
            double num2 = Double.parseDouble(scanner.nextLine());

            double result;

            switch (operator) {
                case "+":
                    result = num1 + num2;
                    break;
                case "-":
                    result = num1 - num2;
                    break;
                case "*":
                    result = num1 * num2;
                    break;
                case "/":
                    result = num1 / num2;
                    break;
                default:
                    throw new IllegalArgumentException("Invalid
operator!");
            }

            System.out.println("Result: " + result);
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid input! Please enter
numeric operands.");
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

2. Write a program that causes the JVM to throw an `OutOfMemoryError` and catches and handles this error.

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

```

    try {
        int arraySize = Integer.MAX_VALUE;
        int[] array = new int[arraySize];
    } catch (OutOfMemoryError e) {
        System.out.println("Error: Out of memory!");
    }
}
}

```

Introducing JavaFx– Java GUI

Lab 11

Objective: Understand the design principles of graphical user interfaces (GUIs) using layout managers to arrange GUI components. Understand basic component of JavaFx and their interaction used in different program of Java, such as Label, Button, Text Box, Combo Box etc.

Theory:

GUI & GUI Components:

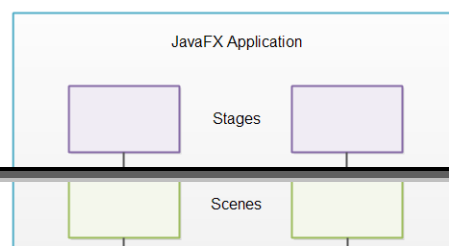
A graphical user interface (GUI) presents a user friendly mechanism for interacting with an application. GUIs are built from GUI components. These are sometimes called controls or widgets. A GUI component is an object with which the user interacts via mouse, the keyboard, or another form of input.

Introduction to JavaFx

JavaFX has replaced Swing as the recommended GUI toolkit for Java. Furthermore, JavaFX is more consistent in its design than Swing, and has more features. It is more modern too, enabling you to design GUI using layout files (XML) and style them with CSS, just like we are used to with web applications. JavaFX also integrates 2D + 3D graphics, charts, audio, video, and embedded web applications into one coherent GUI toolkit.

JavaFx Application Structure

In general, a JavaFX application contains one or more stages which corresponds to windows. Each stage has a scene attached to it. Each scene can have an object graph of controls, layouts etc. attached to it, called the scene graph. These concepts are all explained in more detail later. Here is an illustration of the general structure of a JavaFX application:



Stage: The stage is the outer frame for a JavaFX application. The stage typically corresponds to a window. Each stage is represented by a Stage object inside a JavaFX application. A JavaFX application has a primary Stage object which is created for you by the JavaFX runtime.

Scene: To display anything on a stage in a JavaFX application, you need a scene. A stage can only show one scene at a time, but it is possible to exchange the scene at runtime. A scene is represented by a Scene object inside a JavaFX application. A JavaFX application must create all Scene objects it needs.

Scene Graph: All visual components (controls, layouts etc.) must be attached to a scene to be displayed, and that scene must be attached to a stage for the whole scene to be visible. The total object graph of all the controls, layouts etc. attached to a scene is called the scene graph.

Nodes: All components attached to the scene graph are called nodes. All nodes are subclasses of a JavaFX class called `javafx.scene.Node`.

There are two types of nodes: Branch nodes and leaf nodes. A branch node is a node that can contain other nodes (child nodes). Branch nodes are also referred to as parent nodes because they can contain child nodes. A leaf node is a node which cannot contain other nodes.

JavaFx Controls

JavaFx controls are JavaFx components which provide some kind of control functionality inside a JavaFx application. For instance, a button, radio button, table, tree etc.

For a control to be visible it must be attached to the scene graph of some Scene object.

Controls are usually nested inside some JavaFx layout component that manages the layout of controls relative to each other.

JavaFx contains the following controls:

- Button
- CheckBox
- ColorPicker
- ComboBox
- DatePicker
- Label
- ListView
- Menu

- MenuBar
- PasswordField
- ProgressBar
- RadioButton
- TableView
- TextArea
- TextField

Using different components

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
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.ToggleGroup;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class JavaFxControls extends Application{

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {

        Label heading= new Label("JavaFx Controls");
        heading.setFont(Font.font ("Verdana", 30));
        //heading.setFont(Fon);

        //A button with an empty text caption.
        Button button1 = new Button("Wrong");
        //A button with the specified text caption.
        Button button2 = new Button("Accept");

        // Using radiobuttons
```

```
        RadioButton radioButton1 = new RadioButton("Left");
        RadioButton radioButton2 = new RadioButton("Right");
        RadioButton radioButton3 = new RadioButton("Up");
        RadioButton radioButton4 = new RadioButton("Down");

        // using togglegroup for single selection of radiobuttons
        ToggleGroup radioGroup = new ToggleGroup();

        radioButton1.setToggleGroup(radioGroup);
        radioButton2.setToggleGroup(radioGroup);
        radioButton3.setToggleGroup(radioGroup);
        radioButton4.setToggleGroup(radioGroup);

        // Using CheckBox
        Label l = new Label("What do you listen:      ");
        CheckBox c1 = new CheckBox("Radio one");
        CheckBox c2 = new CheckBox("Radio Mirchi");
        CheckBox c3 = new CheckBox("Red FM");
        CheckBox c4 = new CheckBox("FM GOLD");

        //using ComboBox
        Label l2 = new Label("Where do you live:      ");
        ComboBox comboBox = new ComboBox();

        comboBox.getItems().add("Karachi");
        comboBox.getItems().add("Lahore");
        comboBox.getItems().add("Islamabd");

        // uisng date picker
        Label l3 = new Label("Date : ");
        DatePicker datePicker = new DatePicker();

        // uisng ListView
        Label edu= new Label("Education");
        ObservableList<String> items=
FXCollections.observableArrayList(
        "Phd", "Master", "Graduate", "Intermediate",
        "Matric");

        ListView<String> eduList= new ListView<String>(items);
        eduList.setPrefHeight(40);

        // adding button
        HBox h = new HBox(20);
        h.getChildren().addAll(button1, button2);

        // adding radio button
        HBox h1 = new HBox(10,radioButton1, radioButton2,
radioButton3, radioButton4);

        // adding checkbox
```



```
HBox h2 = new HBox(10);
h2.getChildren().addAll(l,c1,c2,c3,c4);
    // adding Combobox
HBox h3 = new HBox(10);
h3.getChildren().addAll( l2,comboBox);
    //adding date picker
HBox h4 = new HBox(10, l3, datePicker);

//adding listview
HBox h5 = new HBox(10, edu, eduList);

VBox v = new VBox(20);
v.setPadding(new Insets(20));
v.getChildren().addAll(heading, h1, h2, h3, h4, h5, h);

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

stage.setScene(s);
stage.show();
}
}
```

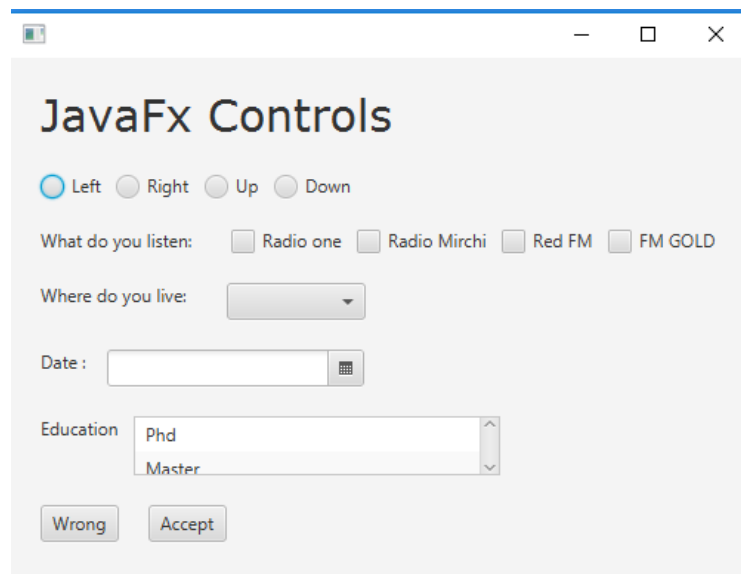


Figure 11.417: JavaFx Controls

Exploring JavaFx

Lab 12

Objective: Show different JavaFx Layouts and Charts.

Theory:

JavaFx Layouts

Layouts are the top level container classes that define the UI styles for scene graph objects. In JavaFX, Layout defines the way in which the components are to be seen on the stage. It basically organizes the scene-graph nodes. We have several built-in layout panes in JavaFX that are HBox, VBox, StackPane, FlowBox, AnchorPane, etc. Each Built-in layout is represented by a separate class which needs to be instantiated in order to implement that particular layout pane.

All these classes belong to **javafx.scene.layout** package. **javafx.scene.layout.Pane** class is the base class for all the built-in layout classes in JavaFX.

Class	Description
BorderPane	Organizes nodes in top, left, right, centre and the bottom of the screen.
FlowPane	Organizes the nodes in the horizontal rows according to the available horizontal spaces. Wraps the nodes to the next line if the horizontal space is less than the total width of the nodes
GridPane	Organizes the nodes in the form of rows and columns.
HBox	Organizes the nodes in a single row.
Pane	It is the base class for all the layout classes.
StackPane	Organizes nodes in the form of a stack i.e. one onto another
VBox	Organizes nodes in a vertical column.

FlowPane

```
import javafx.application.Application;
import javafx.geometry.Orientation;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class FlowPaneExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
```

Submitted To: Abdul Kareem Kashif

ZOHAIB DAYO

```
primaryStage.setTitle("HBox Experiment 1");

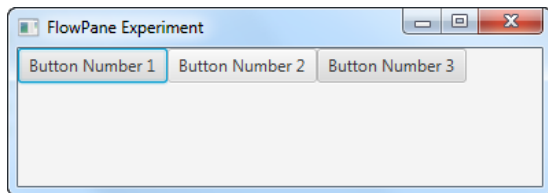
Button button1 = new Button("Button Number 1");
Button button2 = new Button("Button Number 2");
Button button3 = new Button("Button Number 3");

FlowPane flowpane = new FlowPane();

flowpane.getChildren().add(button1);
flowpane.getChildren().add(button2);
flowpane.getChildren().add(button3);

Scene scene = new Scene(flowpane, 200, 100);
primaryStage.setScene(scene);
primaryStage.show();
}

public static void main(String[] args) {
    Application.launch(args);
}
}
```



BorderPane

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.*;
import javafx.stage.Stage;
public class Label_Test extends Application {

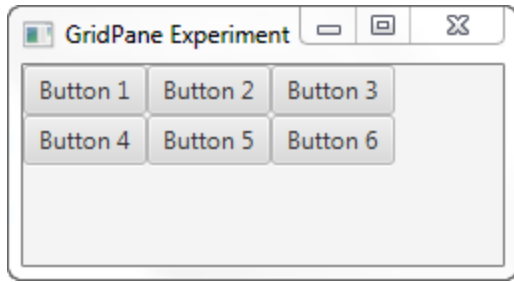
    @Override
    public void start(Stage primaryStage) throws Exception {
        BorderPane BPane = new BorderPane();
        BPane.setTop(new Label("This will be at the top"));
        BPane.setLeft(new Label("This will be at the left"));
        BPane.setRight(new Label("This will be at the Right"));
        BPane.setCenter(new Label("This will be at the Centre"));
        BPane.setBottom(new Label("This will be at the bottom"));
        Scene scene = new Scene(BPane, 600, 400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```
        launch(args);  
    }  
  
}
```

GridPane

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.GridPane;  
import javafx.stage.Stage;  
  
public class GridPaneExperiments extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        primaryStage.setTitle("GridPane Experiment");  
  
        Button button1 = new Button("Button 1");  
        Button button2 = new Button("Button 2");  
        Button button3 = new Button("Button 3");  
        Button button4 = new Button("Button 4");  
        Button button5 = new Button("Button 5");  
        Button button6 = new Button("Button 6");  
  
        GridPane gridPane = new GridPane();  
  
        gridPane.add(button1, 0, 0, 1, 1);  
        gridPane.add(button2, 1, 0, 1, 1);  
        gridPane.add(button3, 2, 0, 1, 1);  
        gridPane.add(button4, 0, 1, 1, 1);  
        gridPane.add(button5, 1, 1, 1, 1);  
        gridPane.add(button6, 2, 1, 1, 1);  
  
        Scene scene = new Scene(gridPane, 240, 100);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        Application.launch(args);  
    }  
}
```



JavaFx Charts

In general, a chart is a graphical representation of data. There are various kinds of charts to represent data such as **Bar Chart, Pie Chart, Line Chart, Scatter Chart**, etc.

JavaFX Provides support for various **Pie Charts** and **XY Charts**. The charts that are represented on an XY-plane include **AreaChart, BarChart, BubbleChart, LineChart, ScatterChart, StackedAreaChart, StackedBarChart**, etc.

Each chart is represented by a class and all these charts belongs to the package **javafx.scene.chart**. The class named **Chart** is the base class of all the charts in JavaFX and the **XYChart** is base class of all those charts that are drawn on the XY-plane.

// Code for Pie Chart

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class PieChartExperiments extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("My First JavaFX App");

        PieChart pieChart = new PieChart();

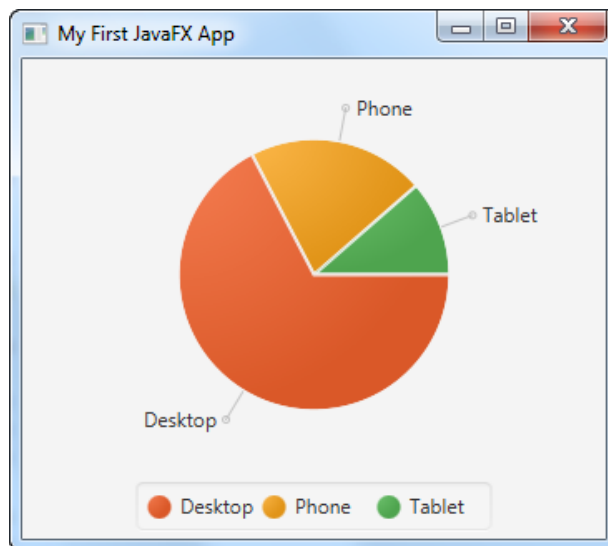
        PieChart.Data slice1 = new PieChart.Data("Desktop", 213);
        PieChart.Data slice2 = new PieChart.Data("Phone", 67);
        PieChart.Data slice3 = new PieChart.Data("Tablet", 36);

        pieChart.getData().add(slice1);
        pieChart.getData().add(slice2);
        pieChart.getData().add(slice3);

        VBox vbox = new VBox(pieChart);

        Scene scene = new Scene(vbox, 400, 200);
```

```
primaryStage.setScene(scene);  
primaryStage.setHeight(300);  
primaryStage.setWidth(1200);  
  
primaryStage.show();  
}  
  
public static void main(String[] args) {  
    Application.launch(args);  
}  
}
```



JavaFx Application

Lab 13

Objective: Design a Login page using JavaFx components.

Login window:

To create a Login Form, we have used two class files:

- NextPage.java
- Login.java

In the Login.java, we have created two text field, text1 and text2 to set the text for username and password. A button is created to perform an action. The method text1.getText() get the text of username and the method text2.getText() get the text of password which the user enters. Then we have create a condition that if the value of text1 and text2 is admin and password respectively, the user will enter into the next page on clicking the submit button. The NextPage.java is created to move the user to the next page. In case if the user enters the invalid username and password, the error message should be displayed.

// code for Login

```
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);

//Add ID's to Nodes
bp.setId("bp");
gridPane.setId("root");
btnLogin.setId("btnLogin");
text.setId("text");

//Action for btnLogin
btnLogin.setOnAction(new
EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        checkUser =
txtUserName.getText().toString();
        checkPw = pf.getText().toString();
        if(checkUser.equals(user) &&
checkPw.equals(pw)) {

            lblMessage.setText("Congratulations!");
            lblMessage.setTextFill(Color.GREEN);
        }
        else{
            lblMessage.setText("Incorrect user or
pw.");
            lblMessage.setTextFill(Color.RED);
        }
        txtUserName.setText("");
        pf.setText("");
    }
}
```

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

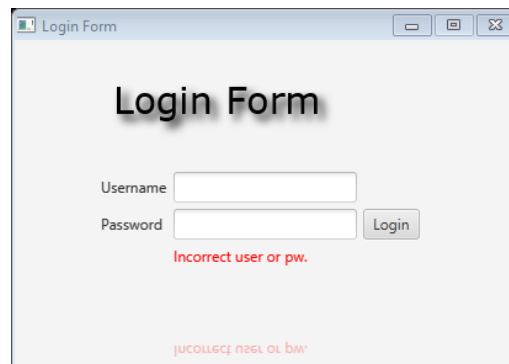


Figure 183.1: Login Screen

//Code For the nextPage.java

```
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 {

    Stage window;
    Scene scene;

    public NextPage() {

        window= new Stage();
        window.setTitle("Add User");
        window.setHeight(370);
        window.setWidth(400);
        window.setResizable(false);

        addcomponents();

        window.setScene(scene);
        window.show();

    }

    private void addcomponents() {
        //Text heading = new Text("Add User");

        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<String>(items);
        eduList.setPrefHeight(40);

        Label loc= new Label("Location");
        ComboBox<String> locList= new ComboBox<String>();
        locList.getItems().add("Karachi");
        locList.getItems().add("Islamabad");
        locList.getItems().add("Multan");
        locList.getItems().add("Lahore");
        locList.getItems().add("Peshawer");

        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);

```

```

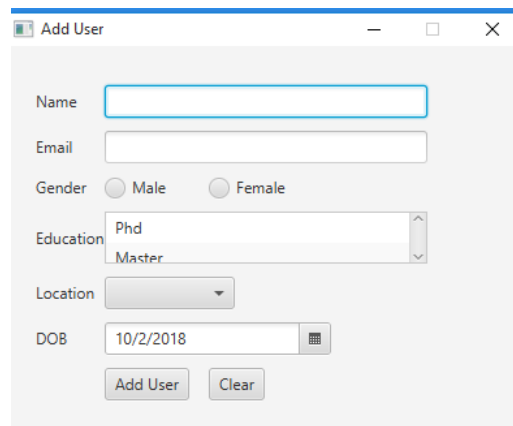
        layout.add(dob, 0, 6);
        layout.add(date, 1, 6);
        layout.add(btnsignup, 1, 7);
        layout.add(btnclear, 1, 7);
        layout.setMargin(btnclear, new Insets(0, 0, 0 ,
80));

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

            @Override
            public void handle(ActionEvent arg0) {
                Alert alert= new
Alert(AlertType.INFORMATION);
                alert.setHeaderText(null);
                alert.setContentText("Added
successfully!!");
                alert.show();
            }
        });

        scene= new Scene(layout);
    }
}

```



Menus

In this section, you will learn about creation of menus, submenus and Separators in JavaFx. Menu bar contains a collection of menus. Each menu can have multiple menu items these are called submenu.

```

import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.control.CheckMenuItem;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;

```

```
import javafx.scene.control.MenuItem;
import javafx.scene.control.RadioMenuItem;
import javafx.scene.control.SeparatorMenuItem;
import javafx.scene.control.ToggleGroup;
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);

        Menu webMenu = new Menu("Web");
        CheckMenuItem htmlMenuItem = new CheckMenuItem("HTML");
        htmlMenuItem.setSelected(true);
        webMenu.getItems().add(htmlMenuItem);

        CheckMenuItem cssMenuItem = new CheckMenuItem("CSS");
        cssMenuItem.setSelected(true);
        webMenu.getItems().add(cssMenuItem);

        Menu sqlMenu = new Menu("SQL");
        ToggleGroup tGroup = new ToggleGroup();
        RadioMenuItem mysqlItem = new RadioMenuItem("MySQL");
        mysqlItem.setToggleGroup(tGroup);

        RadioMenuItem oracleItem = new RadioMenuItem("Oracle");
        oracleItem.setToggleGroup(tGroup);
        oracleItem.setSelected(true);

        sqlMenu.getItems().addAll(mysqlItem, oracleItem,
            new SeparatorMenuItem());

        Menu tutorialMenu = new Menu("Tutorial");
        tutorialMenu.getItems().addAll(
```

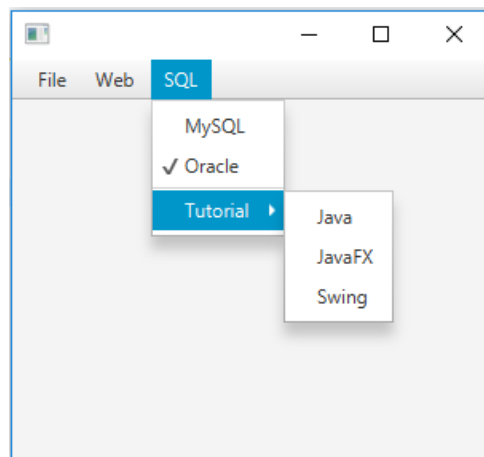
Object Oriented Programming Lab

```
        new CheckMenuItem("Java"),
        new CheckMenuItem("JavaFX"),
        new CheckMenuItem("Swing"));

sqlMenu.getItems().add(tutorialMenu);

menuBar.getMenus().addAll(fileMenu, webMenu, sqlMenu);

primaryStage.setScene(scene);
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```



Data storage and retrieval using filing in Java

Lab 14

Objective: Data storage and retrieval in Signup page using filing.

File Handling in Java

Java File Class: The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

You can read files using these classes:

FileReader for text files in your system's default encoding (for example, files containing Western European characters on a Western European computer).

FileInputStream for binary files and text files that contain 'weird' characters.

You can write files using these classes:

To write a text file in Java, use **FileWriter** instead of FileReader, and **BufferedOutputWriter** instead of BufferedOutputReader.

// Code for data storage and retrieval

```
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
```

```
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<String>(items);
```

```
Label loc= new Label("Location");
ComboBox<String> locList= new ComboBox<String>();
locList.getItems().add("Karachi");
locList.getItems().add("Islamabad");
locList.getItems().add("Multan");
locList.getItems().add("Lahore");
locList.getItems().add("Peshawer");

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();
            }
            //System.out.println(record);

        }
    });

    scene= new Scene(layout);

}

}
```

Introducing UML diagram

Objective: Understand the basics of class diagram of UML.

Theory

UML

The Unified Modeling Language (UML) is a family of graphical notations, backed by single meta model, that help in describing and designing software systems, particularly software systems built using the object-oriented (OO) style.

The UML is a relatively open standard, controlled by the Object Management Group (OMG), an open consortium of companies. The UML was born out of the unification of the many object-oriented graphical modeling languages that thrived in the late 1980s and early 1990s.

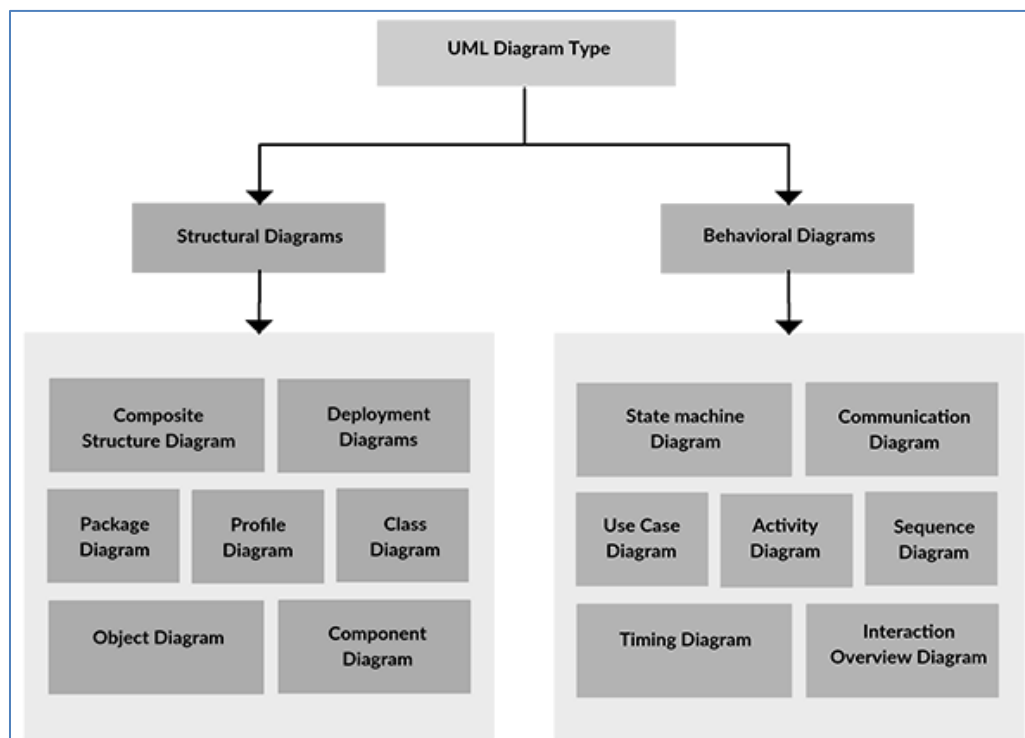


Figure 15.1: UML diagram types

Class Diagram

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them. Class diagrams also show the properties and operations of a class and the constraints that apply to the way objects are connected.

The class diagram is not only widely used but also subject to the greatest range of modeling concepts

Representing a Class

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

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

Attributes are usually listed in the form:

attributeName : Type

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

visibility name (parameters) : return_type

Visibility can be represented as follows

- + public
- # protected
- - private
- / derived



Figure 15.2: class

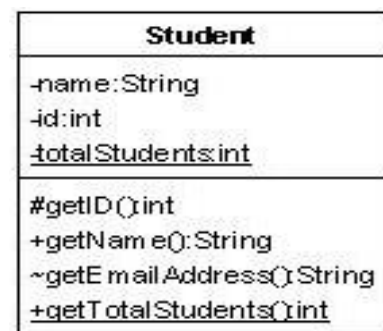


Figure 15.3: Class Example

Relationships in Class Diagram

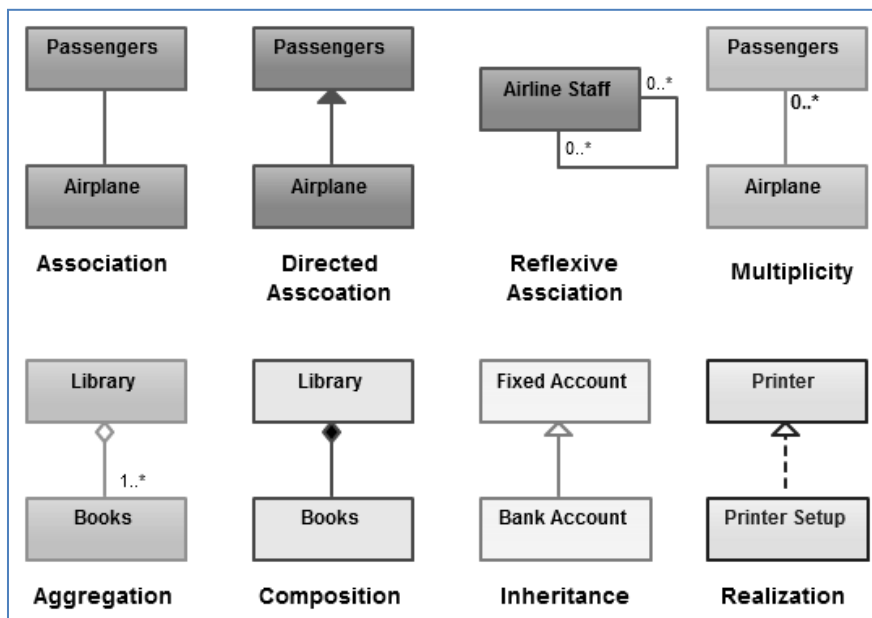


Figure 15.4: Relationships in UML class diagrams

Microsoft Visio

Microsoft Visio is a diagramming and vector graphics application and is part of the Microsoft Office family. The product was first introduced in 1992, made by the Shapeware Corporation. It was acquired by Microsoft in 2000.

Lab Task:

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

Creating Class diagram on Visio

Install the Microsoft Visio on the system. The setup for the software is available on software library. After installation, open Visio. Click on New and then choose option “Software and Database” from the start page.

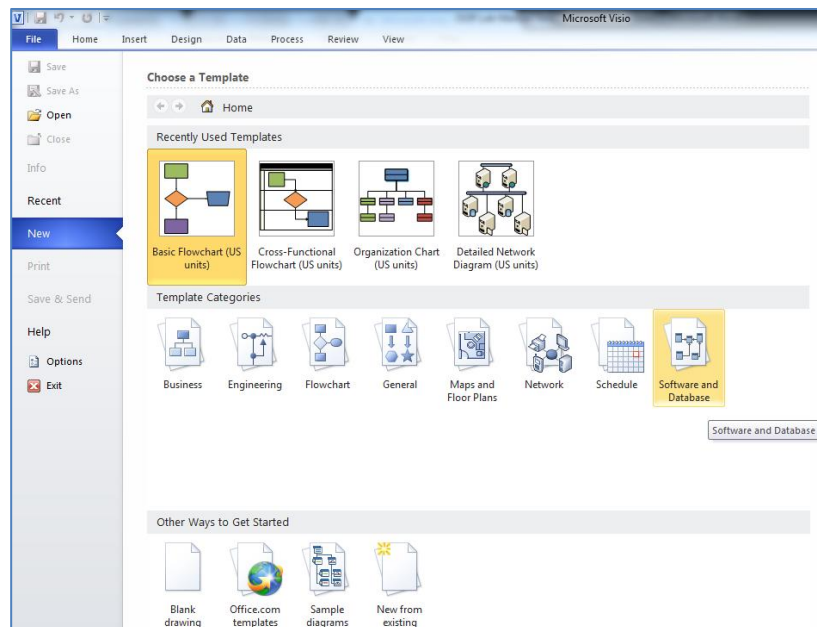
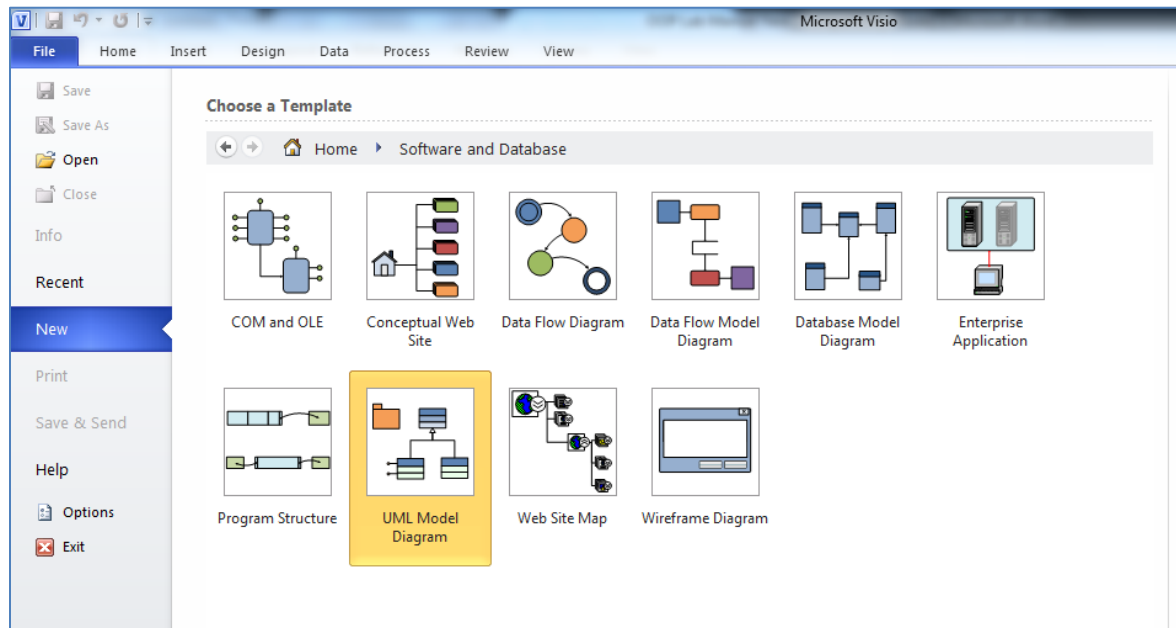


Figure 15.5: Create New Diagram

In the next page, select UML Model Diagram.



All the Class diagram elements are available in the UML Model Diagram template. Drag the element and drop it on the sheet. Use connectors as required.

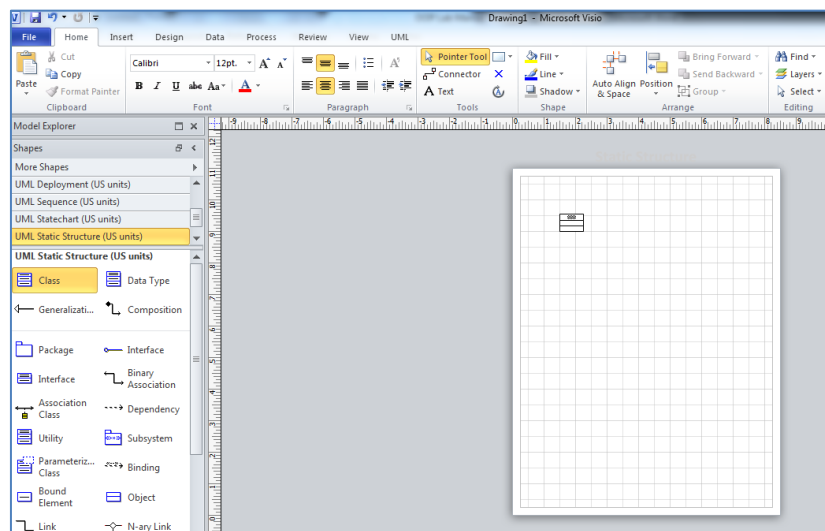


Figure 15.7: Choose Template

Draw the class diagram given below on Visio.

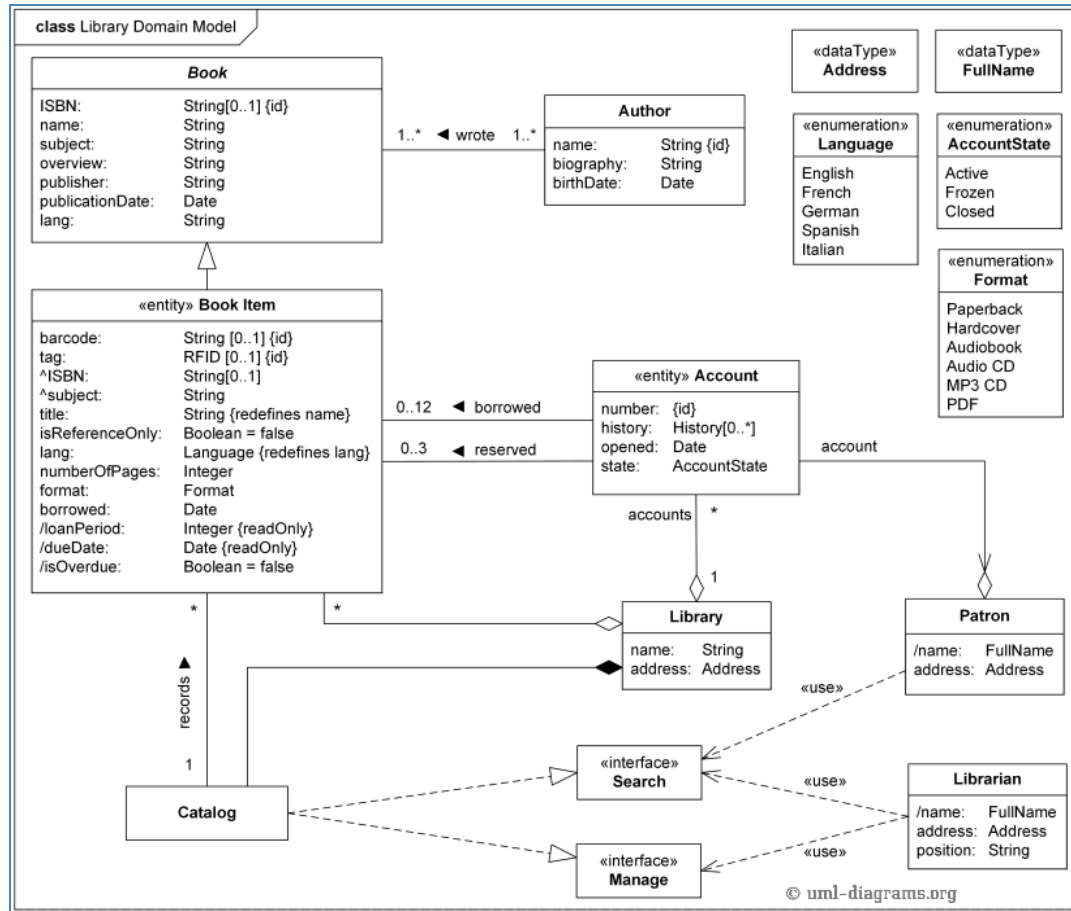


Figure 15.8: Class diagrams for Library Domain Model

Lab Assignment

1. Draw a Class Diagram using MS Visio for the following scenario.

A client wants you to develop a Hospital Management System as per the details provided below.

- There are a lot of people associated with the hospital. Each Person could be associated with different Hospitals, and a Hospital could employ or serve multiple persons.
- Person has derived attributes name and address. Name represents full name and could be combined from title, given (or first) name, middle name, and family (or last) name.
- A Person can be a Patient or Staff. The Staff can be Operations Staff and Administrative Staff. Doctors and Nurses come under Operations Staff.
- Patient can have a unique patient number, date of admission, blood group, sickness, allergies and prescriptions and a derived attribute age which could be calculated based on her or his birth date. A Patient can take medicine, give blood test or X-ray, or do exercise as per instructions.
- Doctor has some specialty, designation and location of his room. A doctor can examine, provide prescription and perform surgery/operation of his patient. A Doctor and a nurse can treat many patients. One patient is treated by 1 or 2 doctors, a senior doctor and a junior doctor.

- The hospital consists of wards. Ward is a division of a hospital or a suite of rooms shared by patients who need a similar kind of care. In a hospital, there are a number of wards, each of which may be empty or have on it one or more patients. Each ward has a unique name and fixed capacity. Wards are differentiated by gender of its patients, i.e. male wards and female wards.

