**Object Oriented Programming**

FALL 22

Semester 3rd

Name: Jaweria

Roll Number: 15787

Department of Computer Science

IQRA University, Karachi

**List of Experiments**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lab No** | **Topics** | **Mapping** | | **Remarks** |
| **PLO 5**  **(5)** | **PLO 9**  **(5)** |
|  | Introduction to Java |  |  |  |
|  | Java Programming Elements |  |  |  |
|  | Java Control Statement, Operators& Arrays |  |  |  |
|  | Introducing Classes, Objects and Strings in java |  |  |  |
|  | Overloading and Access Control |  |  |  |
|  | Inheritance and Polymorphism in Java |  |  |  |
|  | Use of abstract and final |  |  |  |
|  | Packages & Interfaces |  |  |  |
|  | Introducing JavaFx – Java GUI |  |  |  |
|  | Exploring JavaFx |  |  |  |
|  | Exception Handling |  |  |  |
| Total Marks Obtained | |  |  |  |

***Overall Score: \_\_\_\_\_\_\_\_\_ out of 22 Examined by****:* ***Sidra Rehman***

***Overall Formula= (Obtained Score / Total Score)* x *22******(Name and Signature of Concerned)***

**Object Oriented Programming-LAB (3+1)**

### *Instructor: Sidra Rehman*

**Prerequisites:**Computer Programming

**Course Description:**

This course is second programming course for the students who have already taken programming fundamentals / Computer Programming. The course aims to introduce the concepts of object-oriented programming to students with a background in the procedural programming paradigm. The course begins with a brief review of console input/output with emphasis on structured data types and array processing. It then moves on to introduce the object-oriented programming paradigm, focusing on the definition and use of classes along with the fundamentals of object-oriented design including encapsulation and information-hiding; separation of behavior and implementation; classes, subclasses, inheritance; polymorphism; exception handling, visual programming using swing API and event handling.

**Course Contents:**

Evolution of Object Oriented Programming (OOP), Object Oriented concepts and principles, problem solving in Object Oriented paradigm, OOP design process, classes, functions/methods, objects and encapsulation; constructors and destructors, operator and function/method overloading, association, aggregation, composition, generalization, inheritance and its types, derived classes, function/method overriding, abstract and concrete classes, abstract functions, polymorphism, exception handling, Input/output Streams, Swing API and event handling.

**Learning Outcomes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mapping of CLOs and PLOs** | | | |
| **Sr. No.** | **Course Learning Outcomes (CLOs)** | **Program Learning Outcomes (PLOs)** | **Bloom’s Taxonomy** |
| CLO\_1 | Be able to demonstrate the concepts of object-oriented programming principles i.e. encapsulation, abstraction, inheritance and polymorphism. | PLO\_5 | C3 |
| CLO\_2 | Be able to use event handling model to develop event-driven programs that respond to user events. | PLO\_5 | C3 |
| CLO\_3 | Be able to create object oriented solutions for software systems involving fundamental object oriented concepts. | PLO\_5,9 | C3,A2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Assessment tools** | **CLO\_1** | **CLO\_2** | **CLO\_3** |
| Lab Performance/Manual | 25% | 25% | 50% |
| Final Examination | 25% | 25% | 50% |

|  |  |
| --- | --- |
| ***Overall Grading Policy*** | |
| **Assessment Items** | **Percentage** |
| Lab Performance/ Lab Manual | 50% |
| Final Examination | 50% |

**Recommended Book:**

Java How to Program 10th Edition, by Paul Deitel and Harvey Deitel.

**Reference Book:**

Ivor Horton's Beginning Java: Java 7 Edition by Horton, Ivor published by John Wiley & Sons (2011)

**Administrative Instructions:**

* According to institute policy, 75% attendance is *mandatory* to appear in the final examination but 100% will be expected. Approved leaves will not be considered towards attendance.
* Every student should bring manual and notebook in each lab.
* Every student is expected to be in lab before schedule starting time.
* In any case there will be no rescheduling and makeup of labs.

**General Laboratory Procedure**

While there is no specific document to be submitted at the beginning of the Lab –unless your instructor advises you otherwise-, you are expected to read the experiment fully before you come to the laboratory? Interestingly, you can even try parts of the experiment at home. Here is a list of programs that will equip you with a virtual lab at your home:

**Troubleshooting**

Things will not always go as expected; this is the nature of the learning process. While conducting the Experiment **think before you do anything.** If you do so you will avoid wasting time going down dead-end streets. Be logical and systematic. First, look for obvious errors that are easy to fix. Is your measuring device correctly set and connected? Are you looking at the proper scale? Is the power supply set for the correct voltage? Is the signal generator correctly set and connected? How are the variables in the code set? Is there a syntax error? And so on. Next, check for obvious misconnections or broken connections, at least in simple circuits.

As you work through your circuit, use your Lab Manual record tests and changes that you make as you go along; don't rely on your memory for what you have tried. Identify some test points in the system at which you know what the signal should be and work your way backwards from the output through the test points until you find a good signal.

**Neatness**

When you have finished for the day, return all modules to their proper storage bins, return all test leads and probes to their storage racks, return all equipment to its correct location, and clean up the lab station. If appropriate switch off the unneeded equipment. Save your files in the Computer and on any USB device for your records because you might not get the same PC System again for the next experiment. Also email your file contents to your email address as a backup.

**Laboratory Safety**



Always pay attention to what you are doing and you’re surrounding during the experiments, notify the Instructor for any unlikely event or mishap, and leave the Laboratory with the permission of Instructor immediately.

All students must read and understand the information in this document with regard to laboratory safety and emergency procedures prior to the first laboratory session.

**Your personal laboratory safety depends mostly on YOU**. Efforts have been made to address situations that may pose a hazard in the lab but the information and instructions provided cannot be considered all-inclusive.

Students must adhere to written and verbal safety instructions throughout the academic term. Since additional instructions may be given at the beginning of laboratory sessions, it is important that all students arrive at each session on time. With good judgment, the chance of an accident in this course is very small. Nevertheless, research and teaching workplaces (labs, shops, etc.) are full of potential hazards that can cause serious injury and or damage to the equipment. Working alone and unsupervised in laboratories is forbidden if you are working with hazardous substances or equipment. With prior approval, at least two people should be present so that one can shut down equipment and call for help in the event of an emergency. Safety training and/or information should be provided by a faculty member, teaching assistant, lab safety contact, or staff member at the beginning of a new assignment or when a new hazard is introduced into the workplace.

**Emergency Response**

1. It is your responsibility to read safety and fire alarm posters and follow the instructions during an emergency
2. Know the location of the fire extinguisher, eye wash, and safety shower in your lab and know how to use them.
3. Notify your instructor immediately after any injury, fire or explosion, or spill.
4. Know the building evacuation procedures.

**Common Sense**

Good common sense is needed for safety in a laboratory. It is expected that each student will work in a responsible manner and exercise good judgment and common sense. If at any time you are not sure how to handle a particular situation, ask your Teaching Assistant or Instructor for advice **DO NOT TOUCH ANYTHING WITH WHICH YOU ARE NOT COMPLETELY FAMILIAR**!!! It is always better to ask questions than to risk harm to yourself or damage to the equipment.

**Personal and General laboratory safety**

1. Never eat, drink, or smoke while working in the laboratory.
2. Read labels carefully.
3. Do not use any equipment unless you are trained and approved as a user by your supervisor.
4. Wear safety glasses or face shields when working with hazardous materials and/or equipment.
5. Wear gloves when using any hazardous or toxic agent.
6. Clothing: When handling dangerous substances, wear gloves, laboratory coats, and safety shield or glasses. Shorts and sandals should not be worn in the lab at any time. Shoes are required when working in the machine shops.
7. If you have long hair or loose clothes, make sure it is tied back or confined.
8. Keep the work area clear of all materials except those needed for your work. Coats should be hung in the hall or placed in a locker. Extra books, purses, etc. should be kept away from equipment that requires air flow or ventilation to prevent overheating.
9. Disposal - Students are responsible for the proper disposal of used material if any in appropriate containers.
10. Equipment Failure - If a piece of equipment fails while being used, report it immediately to your lab assistant or tutor. Never try to fix the problem yourself because you could harm yourself and others.
11. If leaving a lab unattended, turn off all ignition sources and lock the doors.
12. Never pipette anything by mouth.
13. Clean up your work area before leaving.
14. Wash hands before leaving the lab and before eating.
15. Unauthorized person(s) shall not be allowed in a laboratory for any reason

**Electrical safety**

1. Obtain permission before operating any high voltage equipment.
2. Maintain an unobstructed access to all electrical panels.
3. Wiring or other electrical modifications must be referred to the Electronics Shop or the Building Coordinator.
4. Avoid using extension cords whenever possible. If you must use one, obtain a heavy- duty one that is electrically grounded, with its own fuse, and install it safely. Extension cords should not go under doors, across aisles, be hung from the ceiling, or plugged into other extension cords.
5. Never, ever modify, attach or otherwise change any high voltage equipment.
6. Always make sure all capacitors are discharged (using a grounded cable with an insulating handle) before touching high voltage leads or the "inside" of any equipment even after it has been turned off. Capacitors can hold charge for many hours after the equipment has been turned off.
7. When you are adjusting any high voltage equipment or a laser which is powered with a high voltage supply, USE ONLY ONE HAND. Your other hand is best placed in a pocket or behind your back. This procedure eliminates the possibility of an accident where high voltage current flows up one arm, through your chest, and down the other arm.
8. Discard damaged cords, cords that become hot, or cords with exposed wiring.
9. Before equipment is energized ensure, (1) circuit connections and layout have been checked by a Teaching Assistant (TA) and (2) all colleagues in your group give their assent.
10. Know the correct handling, storage and disposal procedures for batteries, cells, capacitors, inductors and other high energy-storage devices.
11. Experiments left unattended should be isolated from the power supplies. If for a special reason, it must be left on, a barrier and a warning notice are required.
12. Equipment found to be faulty in any way should be reported to the Lab Engineer immediately and taken out of service until inspected and declared safe.
13. Voltages above 50 V rms AC and 120 V DC are always dangerous. Extra precautions should be considered as voltage levels are increased.
14. Never make any changes to circuits or mechanical layout without first isolating the circuit by switching off and removing connections to power supplies.
15. Know what you must do in an emergency.
16. Emergency Power Off: Every lab is equipped with and Emergency Power Off System.
17. Only authorized personnel are permitted to reset power once the Emergency Power Off system has been engaged.

**Electrical Emergency Response**

The following instructions provide guidelines for handling two types of electrical emergencies:

1. When someone suffers serious electrical shock, he or she may be knocked unconscious. If the victim is still in contact with the electrical current, immediately turn off the electrical power source. If you cannot disconnect the power source, depress the Emergency Power Off switch.
2. Do not touch a victim that is still in contact with a live power source; you could be electrocuted.
3. Have someone call for emergency medical assistance immediately. Administer first-aid, as appropriate.
4. If an electrical fire occurs, try to disconnect the electrical power source, if possible. If the fire is small and you are not in immediate danger; and you have been properly trained in fighting fires, use the correct type of fire extinguisher to extinguish the fire. When in doubt, push in the Emergency Power Off button.
5. NEVER use water to extinguish an electrical fire.

**Mechanical safety**

1. When using compressed air, use only approved nozzles and never direct the air towards any person.
2. Guards on machinery must be in place during operation.
3. Exercise care when working with or near hydraulically- or pneumatically-driven equipment. Sudden or unexpected motion can inflict serious injury.

**Additional Safety Guidelines**

1. Never do unauthorized experiments.
2. Never work alone in laboratory.
3. Keep your lab space clean and organized.
4. Do not leave an on-going experiment unattended.
5. Always inform your instructor if you break a thermometer. Do not clean mercury yourself!!
6. Never taste anything. Never pipette by mouth; use a bulb.
7. Never use open flames in laboratory unless instructed by TA.
8. Check your glassware for cracks and chips each time you use it. Cracks could cause the glassware to fail during use and cause serious injury to you or lab mates.
9. Maintain unobstructed access to all exits, fire extinguishers, electrical panels, emergency showers, and eye washes.
10. Do not use corridors for storage or work areas.
11. Do not store heavy items above table height. Any overhead storage of supplies on top of cabinets should be limited to lightweight items only. Also, remember that a 36" diameter area around all fire sprinkler heads must be kept clear at all times.
12. Areas containing lasers, biohazards, radioisotopes, and carcinogens should be posted accordingly. However, do not post areas unnecessarily and be sure that the labels are removed when the hazards are no longer present.
13. Be careful when lifting heavy objects. Only shop staff may operate forklifts or cranes.
14. Clean your lab bench and equipment, and lock the door before you leave the laboratory.

**Clothing**

1. Dress properly during a laboratory activity.
2. Long hair, dangling jewelry, and loose or baggy clothing are a hazard in the laboratory.
3. Long hair must be tied back, and dangling jewelry and baggy clothing must be secured.
4. Shoes must completely cover the foot.
5.  No sandals allowed on lab days.
6. A lab coat or smock should be worn during laboratory experiments.

**Accidents and Injuries**

1. Do not panic.
2. Report any accident (spill, breakage, etc.) or injury (cut, burn, etc.) to the teacher immediately, no matter how trivial it seems.
3. If you or your lab partner is hurt, immediately (and loudly) yell out the teacher's name to get the teacher's attention.

**General Warning Signs**



**Lab-01**

**Introduction to Java**

## Objectives:

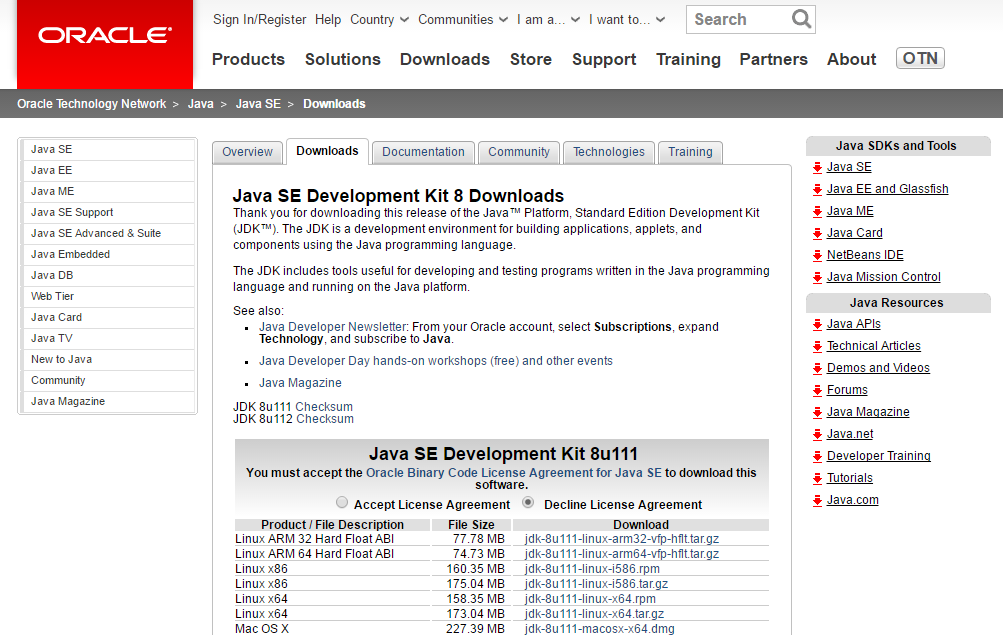
Getting familiar with the Java development kit (JDK). Running your first Java program using CMD and an IDE.

**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.

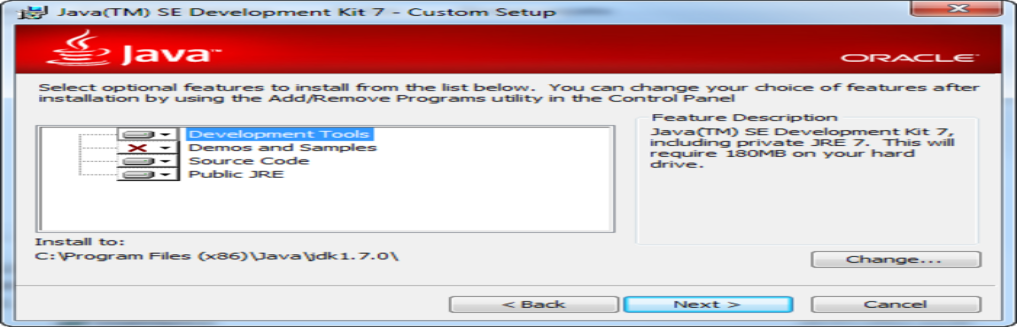
**Installing JDK for Windows**

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

****

**Figure 1.1: Download the JDK from Oracle’s website**

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



**Figure 1.2 Installing JDK**

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



**Figure 1.3: Successful installation**

1. Post installation, you will need to set some environment variables in windows.
2. 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
3. **How to set Temporary Path of JDK in Windows**

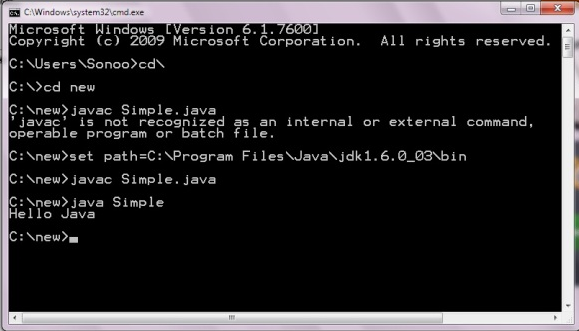
To set the temporary path of JDK, you need to follow following steps:

* Open command prompt
* Copy the path of jdk/bin directory
* Design 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:

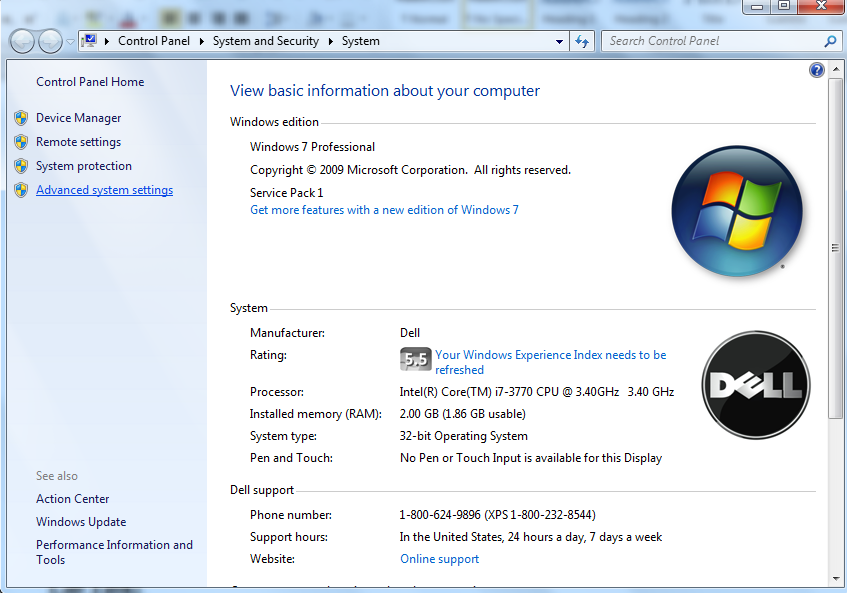


**Figure 1.4: Command Prompt**

1. **How to set Permanent Path of JDK in Windows**

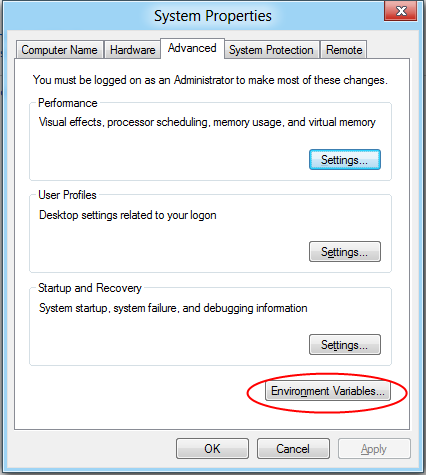
For setting the permanent path of JDK, you need to follow these steps:

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



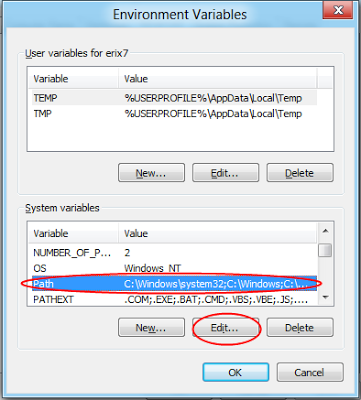
**Figure 1.5: Advance System Settings - Windows**

* Click on environment variables



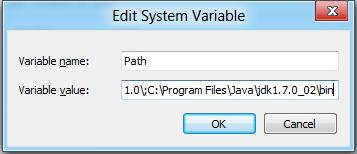
**Figure 1.6: Set Environment Variables**

* 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:**

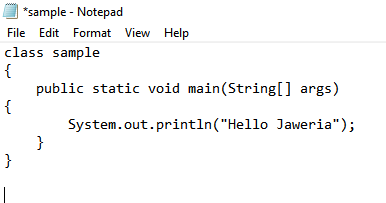
The PATH environment variable is a series of directories separated by semicolons (;) and is not case-sensitive. Microsoft Windows looks for programs in the PATH directories in order, from left to right.

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

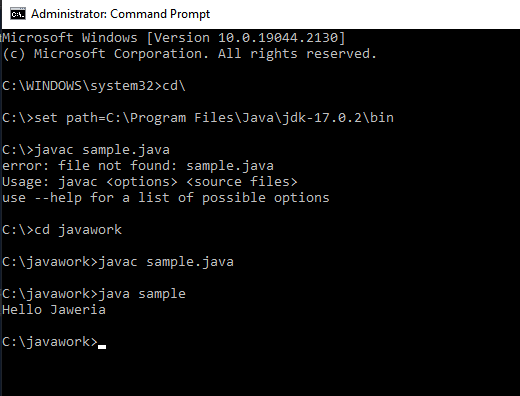
**Lab Task:**

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

***Code***



***Output***



## Conclusion:

In this lab we learn about basic programing and how to use command

**What is an IDE?**

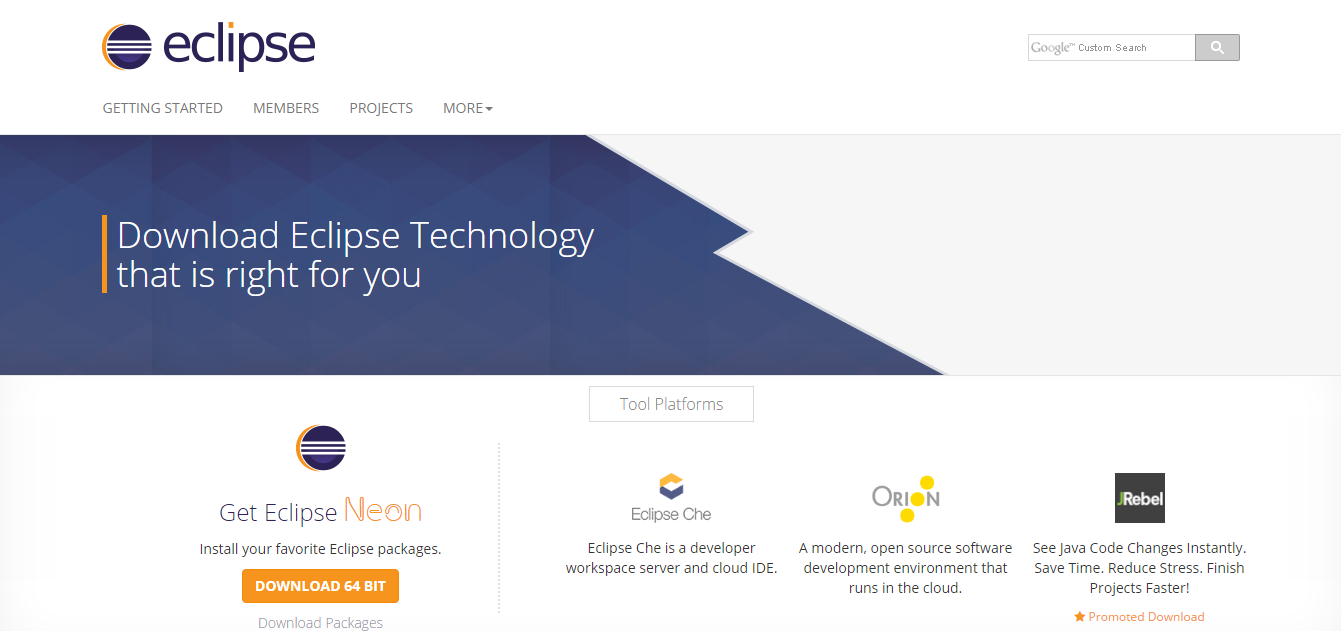
An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE may consist of a source code editor; build automation tools, compiler and a debugger.

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.

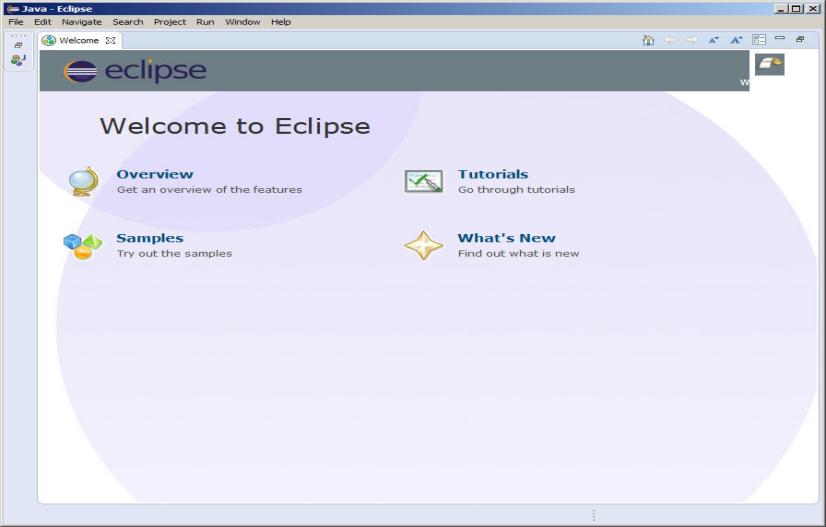
**Install Eclipse IDE**

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



**Figure 1.9: Download Eclipse IDE**

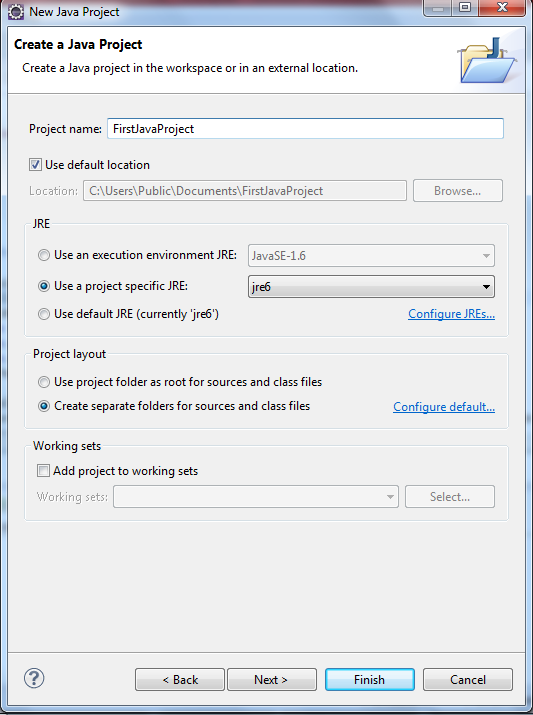
After installation, open eclipse. You will see the welcome screen.



**Figure 1.10: Welcome Screen of Eclipse**

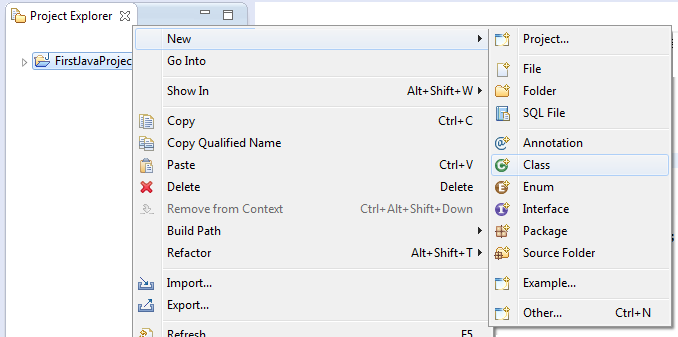
**Design a simple program in java, compile and run it using IDE.**

* 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.



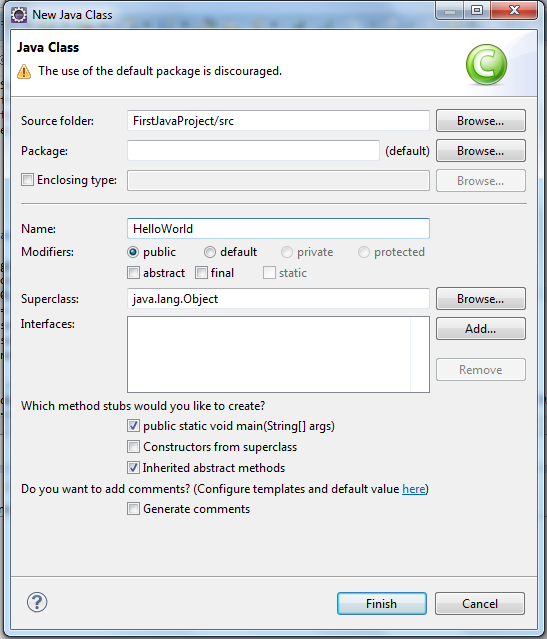
**Figure 1.11: Creating new project in Eclipse**

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



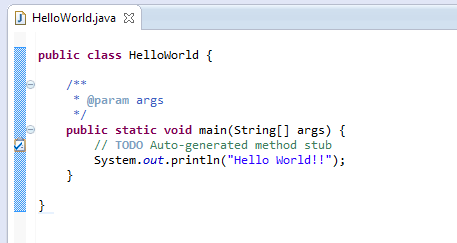
**Figure 1.12: Add new Class**

* 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.



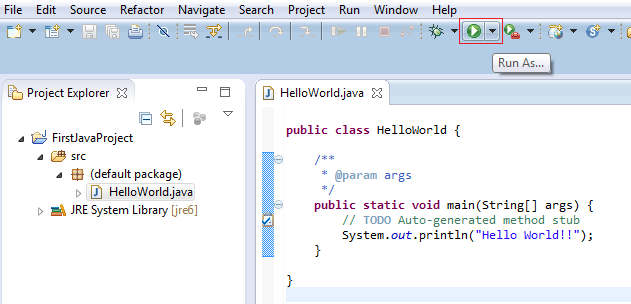
**Figure 1.13: Creating new Class**

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



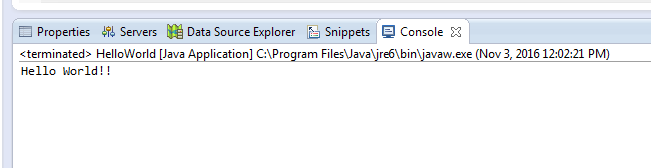
**Figure 1.14: Adding Java Code**

* 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.

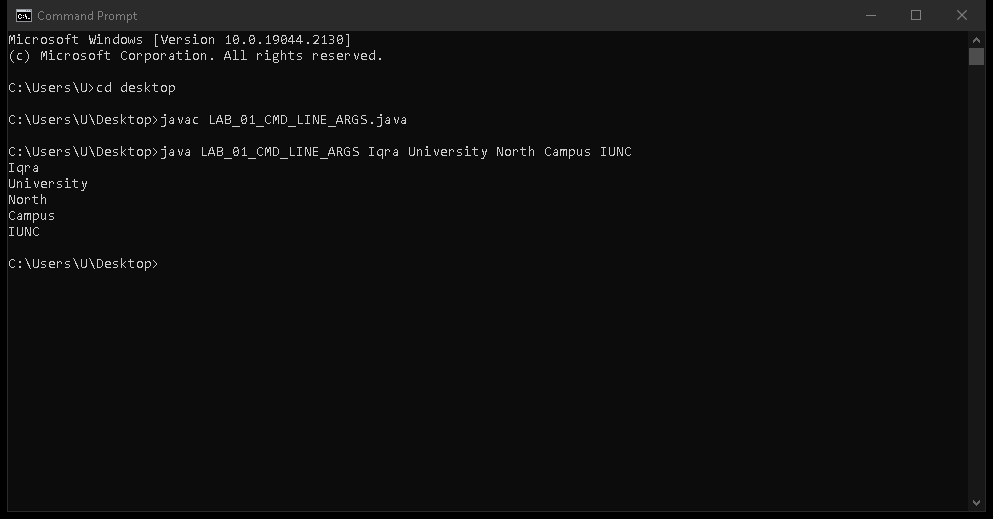


**Figure 1.16: Console Output**

**Lab Assignment:**

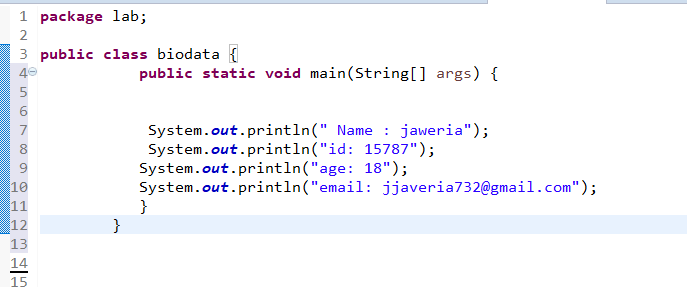
* 1. **Design a simple java program with command-line argument in java.**

***Output***

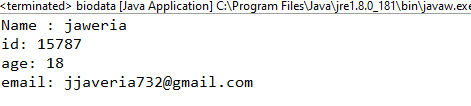
****

* 1. **Design a java program to print your bio data using java.**

***Code***



**Output**

****

## Conclusion:

In this lab we learn about basic programing and how to use libraries, how to create class and how to print your bio data in java programming language

**Lab-02**

**Java Programming Elements**

## Objectives:

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 %.2f", total);

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

**Data Types in Java**

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

There are two basic types in Java.

1. **Primitive**  
    A primitive type is predefined by the language and is named by a reserved keyword.
2. **Non-Primitive**

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

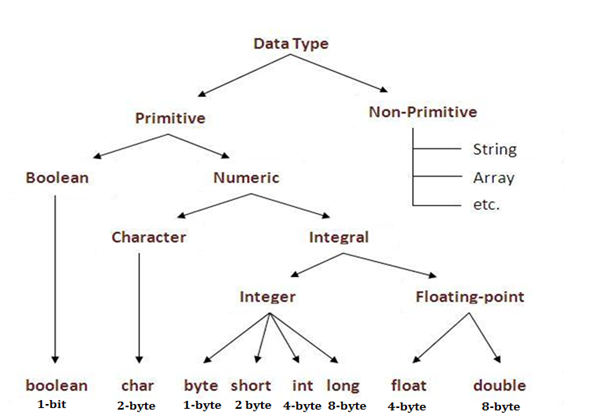


Figure 2.1: Data Types in Java

**Variables**

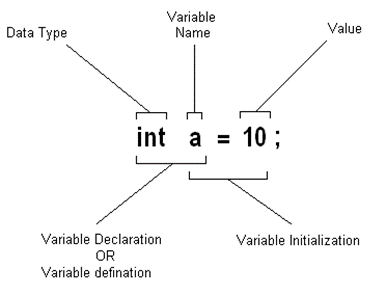
****

Figure 2.2: Variable Initialization

* 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;

**Type Conversion**

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

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

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

**Some Useful Java Classes**

**Math**

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

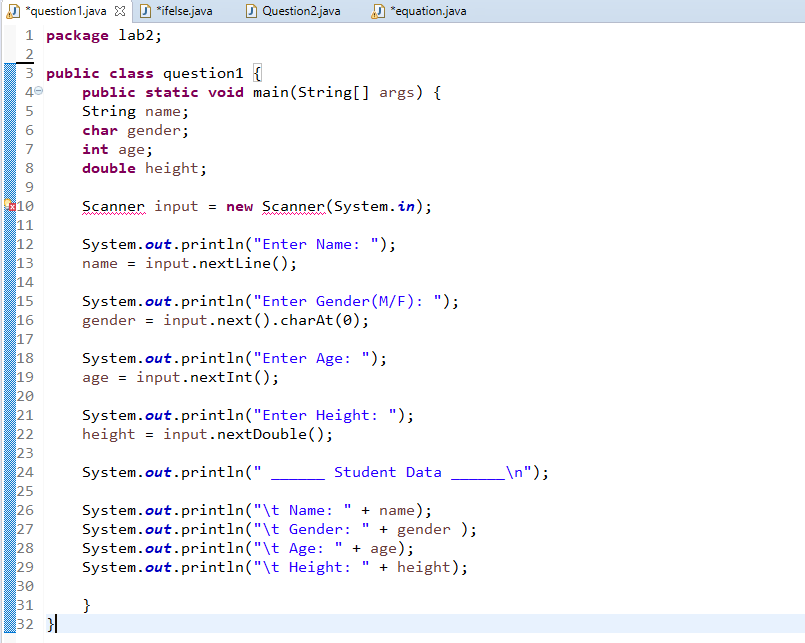
Trignometic / Maths Functions

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

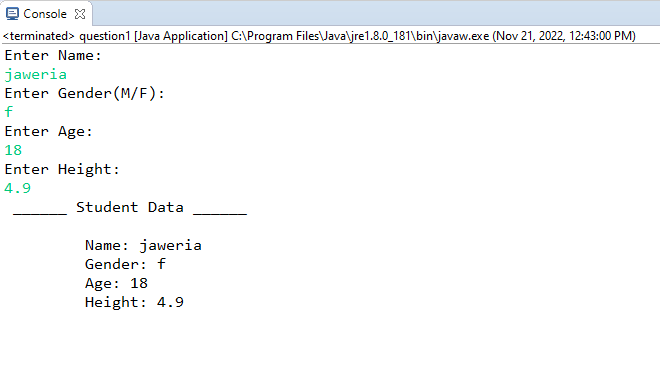
**Lab Task:**

1. Design 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.

***Code***

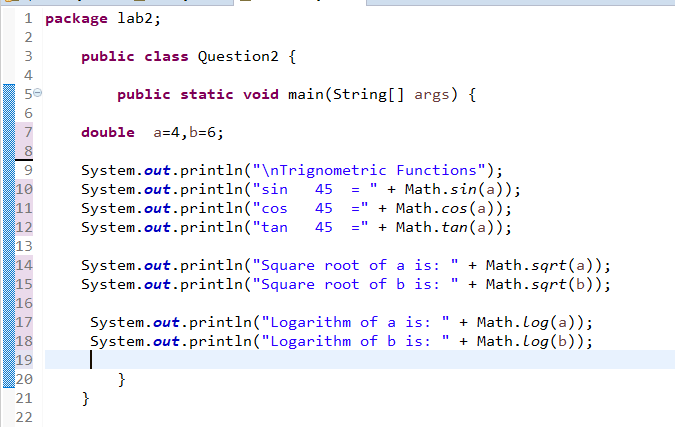
****

***Output***

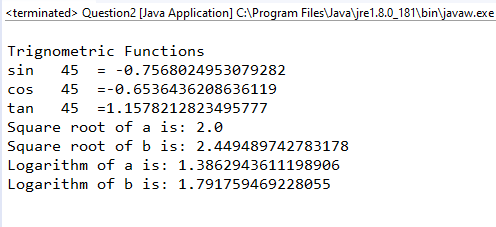
******

1. Design a Java program to explore Math class.

***Code***



***Output***

***z***

**Lab Assignment:**

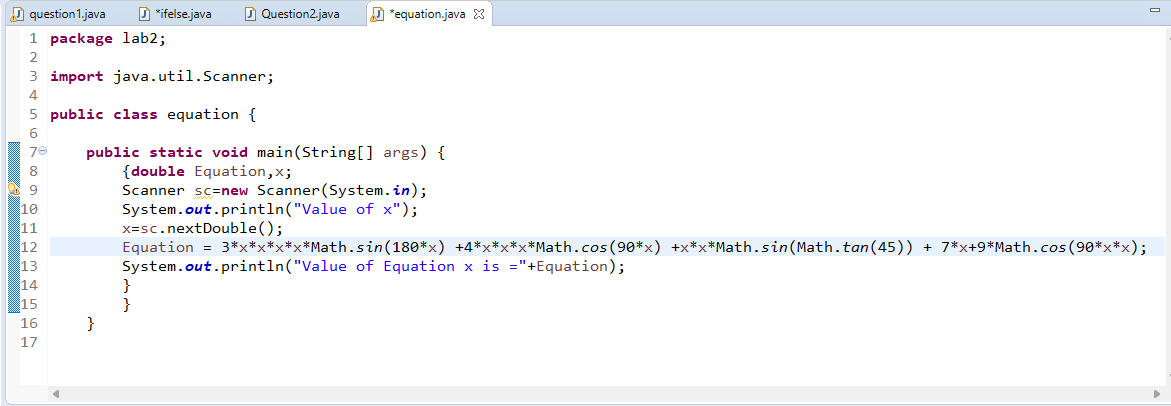
**Program the following.**

Implement the following equation

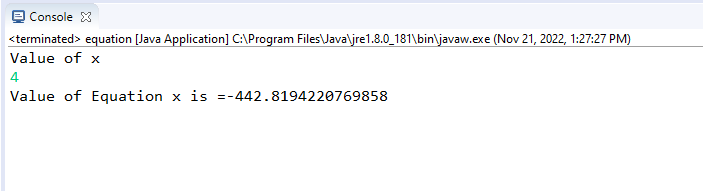
3x4sin(180x) + 4x3cos(90x) + x2sin(tan(45)) + 7x + 9cos(90x2 )

Where x may be user defined value

***Input***

****

***Output***

****

**Conclusion:**

In this lab we learn about how to use math libraries and how to input in user and print in java programming language

**Lab-03**

**Classes, Objects & Strings**

**Objectives:**

Understanding concepts of class and object in java. Implementing a class with members including data, methods and constructors also getting familiar with the String class of Java.

**Theory:**

**Class**

A class consists of

* Data(variables)
* Methods
* Constructors

**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 manylanguages, strings are treated as an array of characters, but in Java a string is treated as anobject.A String object is immutable; its contents cannot be changed.

**Syntax:**

String newString = new String(stringLiteral);

String newString = stringLiteral;

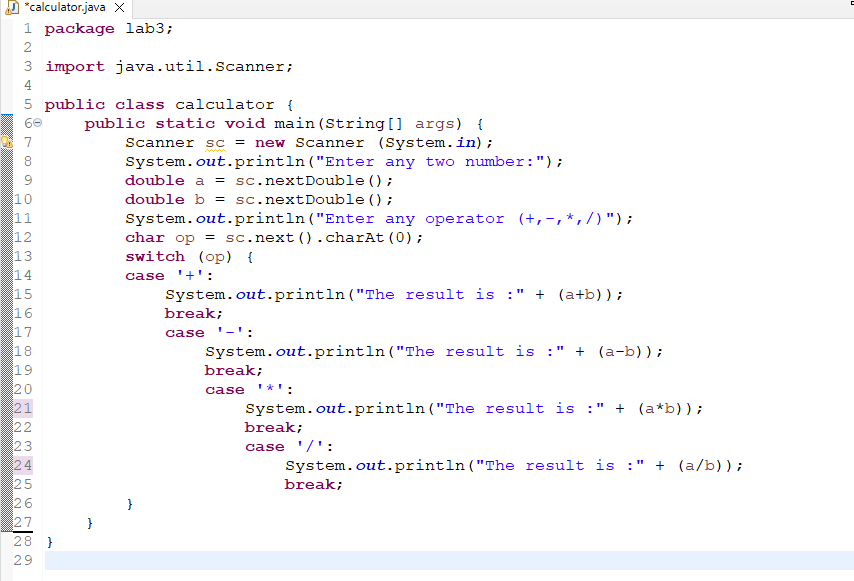
The String class provides the methods for comparing strings.

|  |  |
| --- | --- |
| 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 |

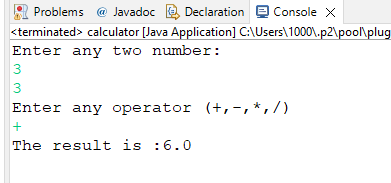
**Lab Assignment:**

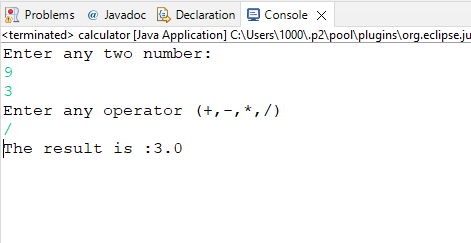
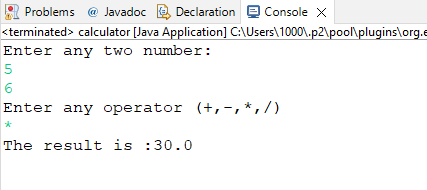
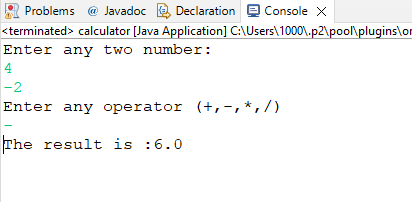
1. **Create a class Calculator and implement all the basic operations for two objects**

***Code***

******

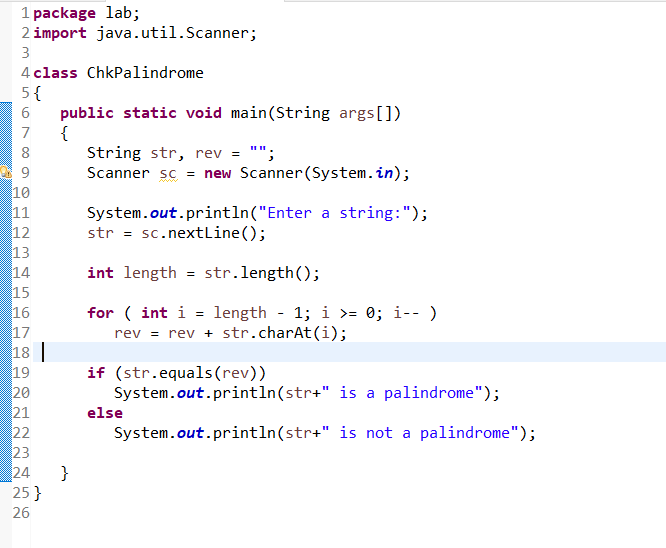
***Output***

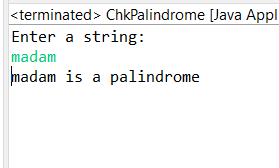
******

******

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

***Code***

 ***Output***



**Conclusion:**

In this lab we learn abouthow to use if else statement and how to program a basic calculator in java programming language

**Lab-04**

**Java Control Statement, Operators And Arrays.**

**Objectives:**

Understanding the control statements of Java including Loops & if-else.Exploring different operators used in Java and Arrays, array index, single and multi-dimensional arrays, traversing the array using loop

**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**

|  |  |  |
| --- | --- | --- |
| ***while*** | ***do-while*** | ***for*** |
| while (loop-continuation-condition)  {  // Loop body Statement(s);  } | do {  // Loop body; Statement(s);  }  while (loop-continuation-condition); | for (initial-action;  loop-continuation-condition;  action-after-each-iteration)  {  // Loop body; Statement(s);  } |

**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**

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

***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.

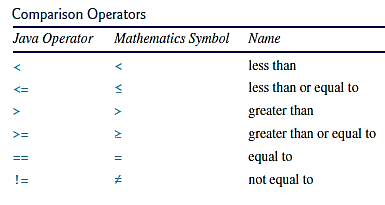


Figure 2.3: Comparison Operators

**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.

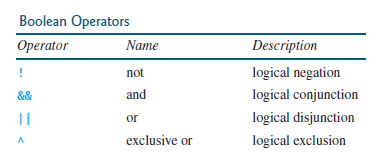


Figure 2.3: Boolean Operators

**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:

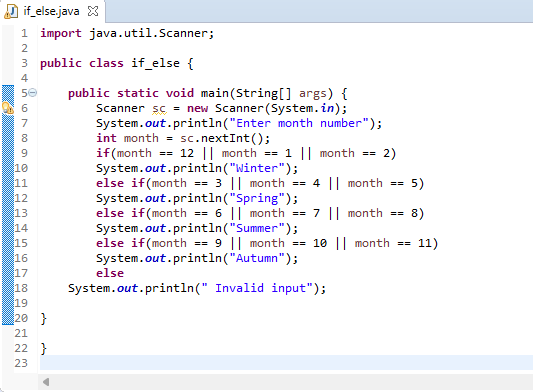
arrayRefVar[index] = value; // 1d array

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

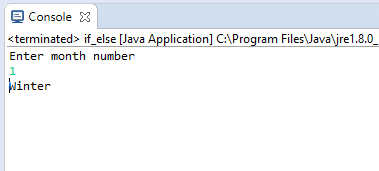
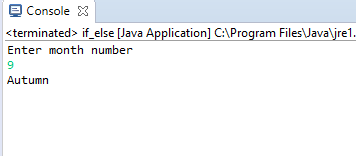
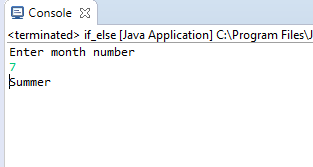
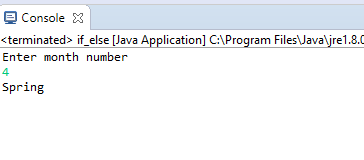
**Lab Task:**

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

***Code***

******

***Output***

******

1. **Design a Java program to generate following pattern**

**.....**

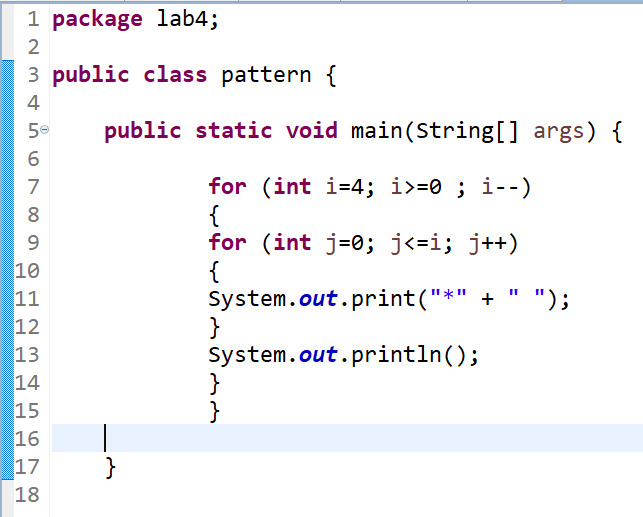
**....**

**...**

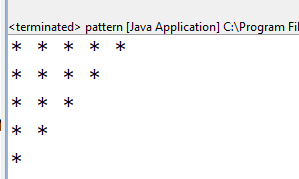
**..**

**.**

***Code***

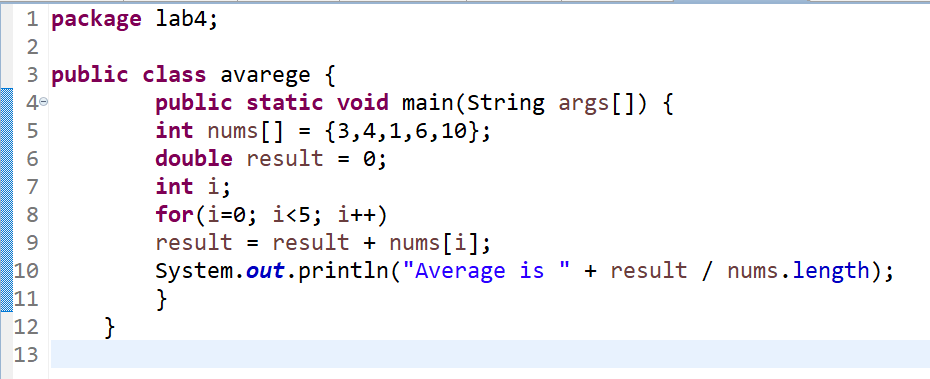


***Output***

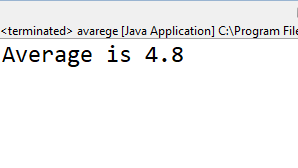


1. **Design a program using one-dimensional array of numbers and finds the average of a set of numbers.**

***Code***

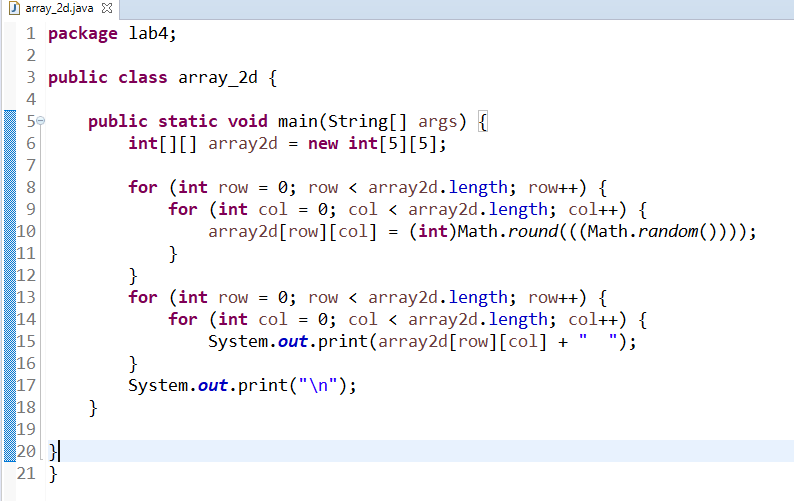


***Output***

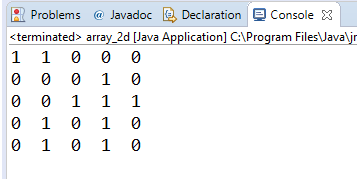


1. **Design a program to initialize the elements of 2D array with random numbers.**

***Code***

****

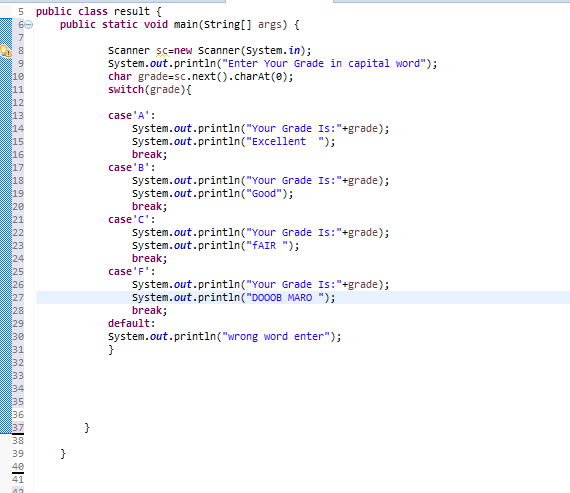
**Output**

****

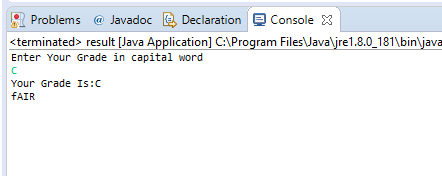
**Lab Assignment:**

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

***Code***

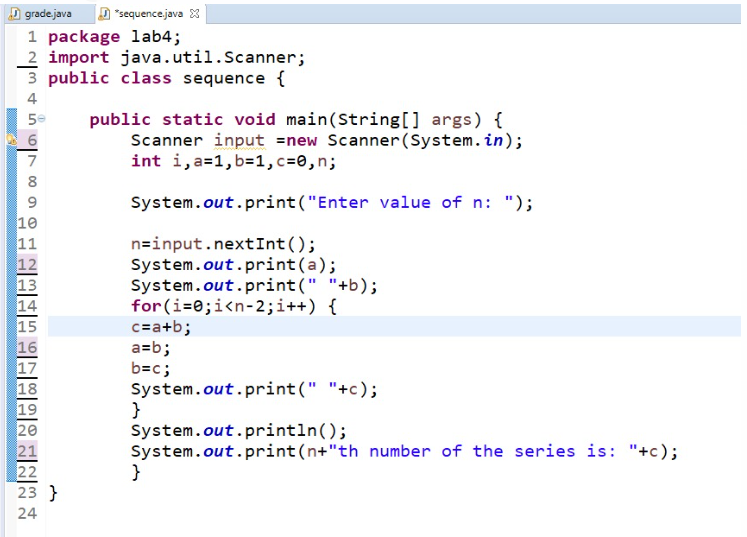
****

**Output**

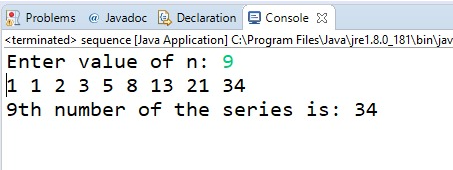
****

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

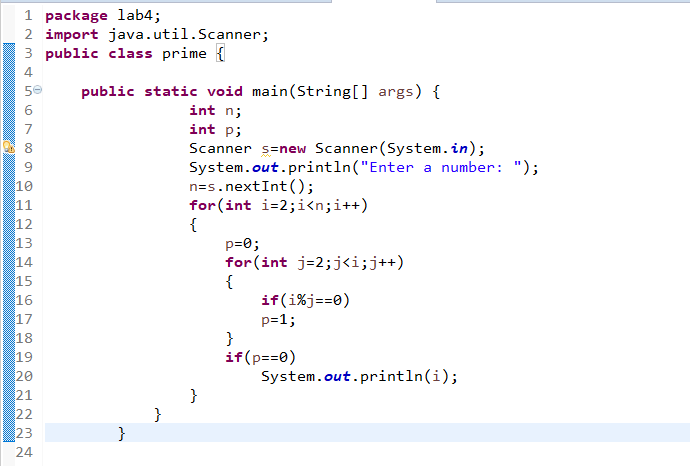
***Code***

******

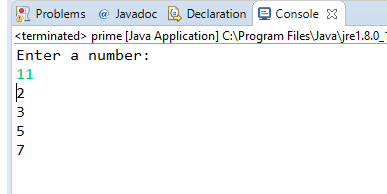
***Output***

******

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

***Code ***

***Output***

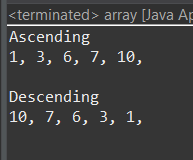


1. **Design a Java program for sorting a given array of numbers in ascending order and descending order?**

***Code***

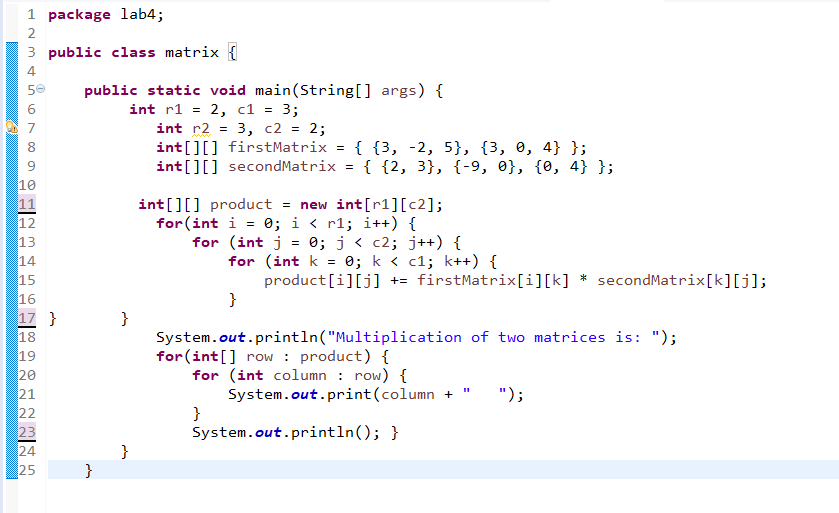
******

***Output***

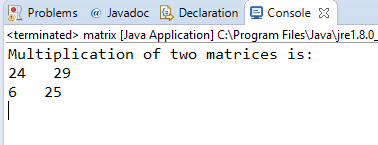
******

1. **Design a Java program to multiply two given matrices using 2D array.**

***Code***

******

***Output***

****

**Conclusion:**

In this lab we learn about switch case, loops statements and arrays

**Java provides three types of Loops: for, while, and do-while**.

**java provides switch and multiple cases**

**java provide array and 2d array**

**Lab-05**

**Overloading & Access Control**

**Objectives:**

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.

Three ways to overload a method

In order to overload a method, the parameter list of the methods must differ in either of these:

**1. Number of parameters.**

For example: This is a valid case of overloading

add(int, int)

add(int, int, int)

**2. Data type of parameters.**

For example:

add(int, int)

add(int, float)

**3. Sequence of Data type of parameters.**

For example:

add(int, float)

add(float, int)

**Invalid case of method overloading:**

Parameters list doesn’t mean the return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw a compilation error.

int add(int, int)

float add(int, int)

**Type Promotion table:**

The data type on the left side can be promoted to any of the data type present at the right side.

byte → short → int → long → double

short → int → long → float → double

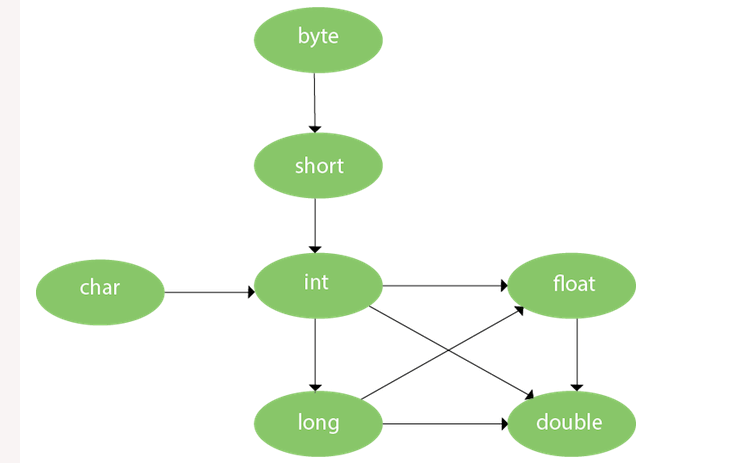
int → long → float → double

float → double

long → float → double

char → int → long → float → double

This can be represented as a diagram like this:



* **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.

classOverloadDemo {

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);

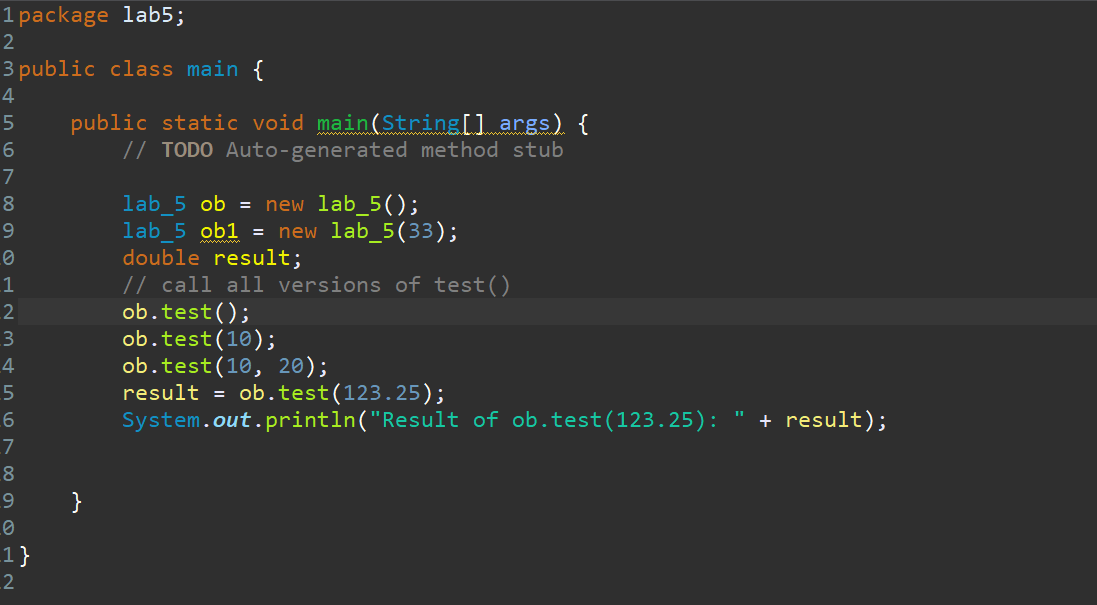
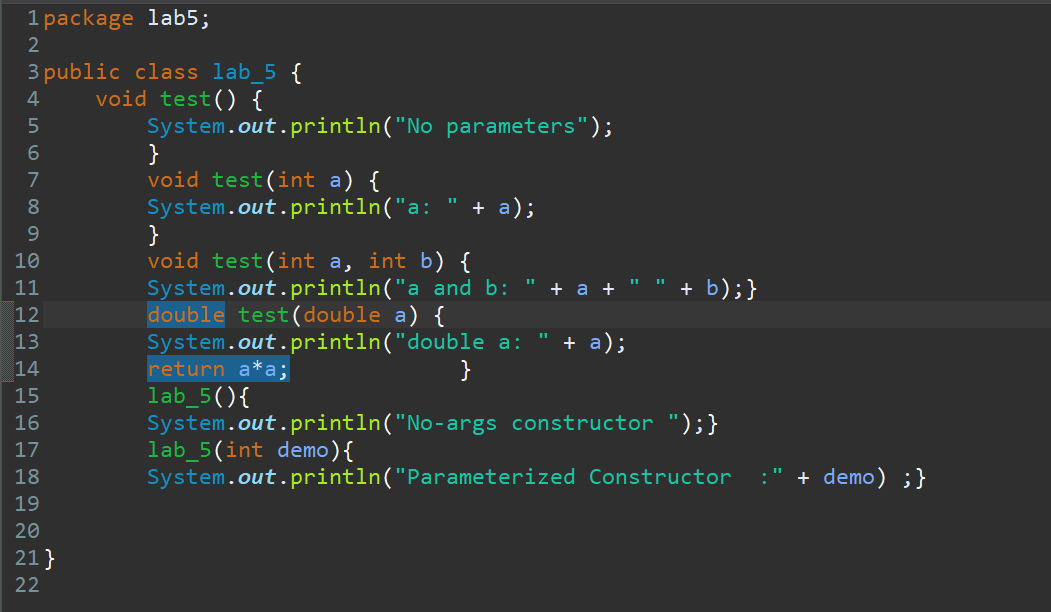
result = ob.test(123.25);

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

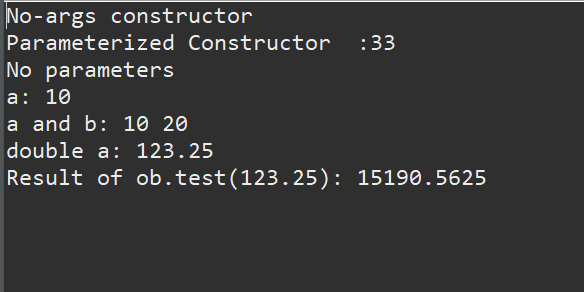
}

}

**code**



**Output**



**Lab Assignment:**

**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.

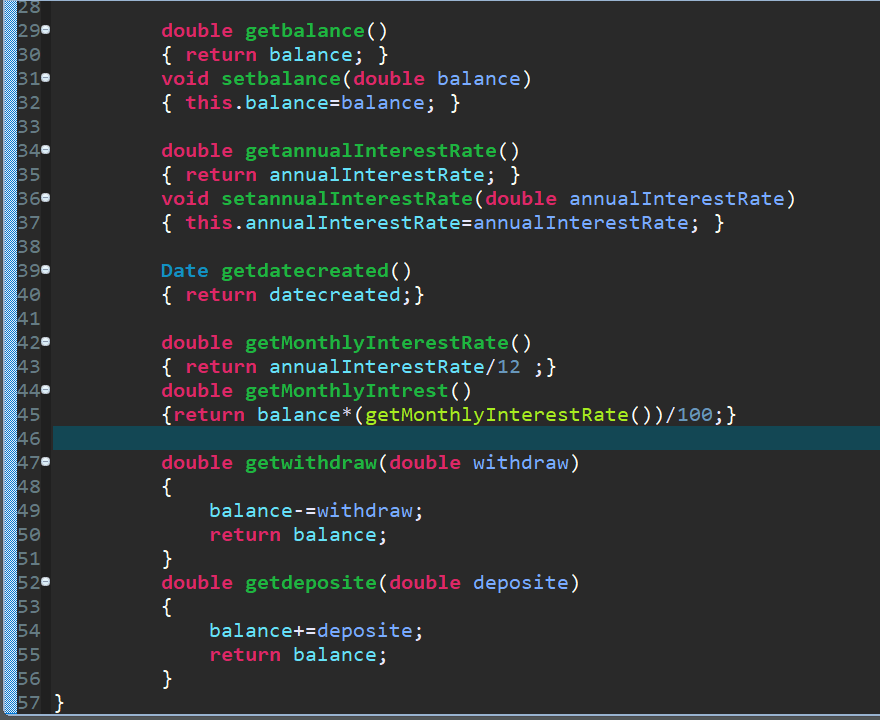
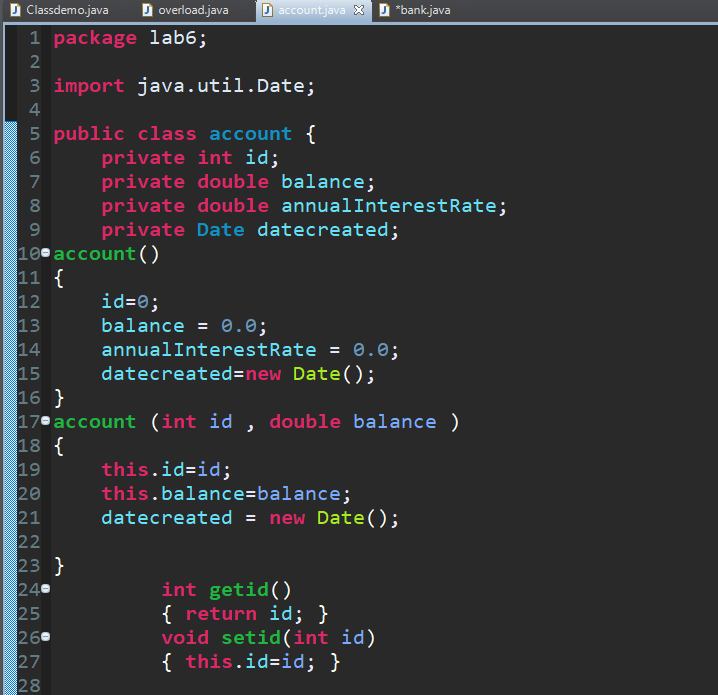
monthlyInterestRate is annualInterestRate / 12.

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

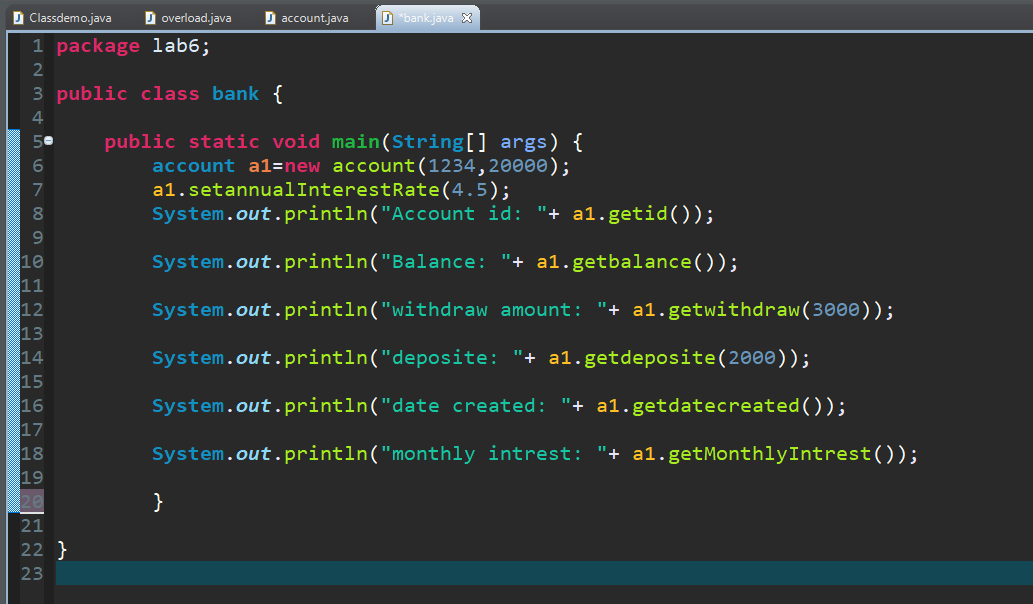
Design a test program that creates an Account object with an account ID of 1122, a balance of $20,000, and an annual interest rate of 4.5%. Use the withdraw method to withdraw $2,500, use the deposit method to deposit $3,000, and print the balance, the monthly interest, and the date when this account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

**Code**

**(sub class)**

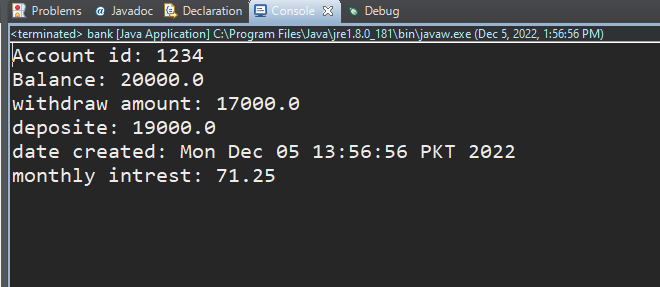
 }

(main class)



}

**Output**

****

**Conclusion:**

In this lab we learn abouthow to use overloading and access control in java programming language

**Lab-06**

**Inheritance and Polymorphism in Java**

**Objectives:**

**Understanding the concept of inheritance, the superclass and subclass and polymorphism**

**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.

**Polymorphism**

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

**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**

//edit class BoxWeight

class BoxWeight extends Box {

double weight;

// construct clone of an object

BoxWeight(BoxWeight ob) { // pass object to constructor

super(ob);

weight = ob.weight;

}

// constructor used when all dimensions specified

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

{

super (w,h,d);

weight = m;

}

// constructor used when no dimensions specified

BoxWeight() {

super();

weight = -1; // use -1 to indicate an uninitialized box

}

// constructor used when cube is created

BoxWeight(double len,double m) {

super(len);

weight=m;

}

}

// 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);

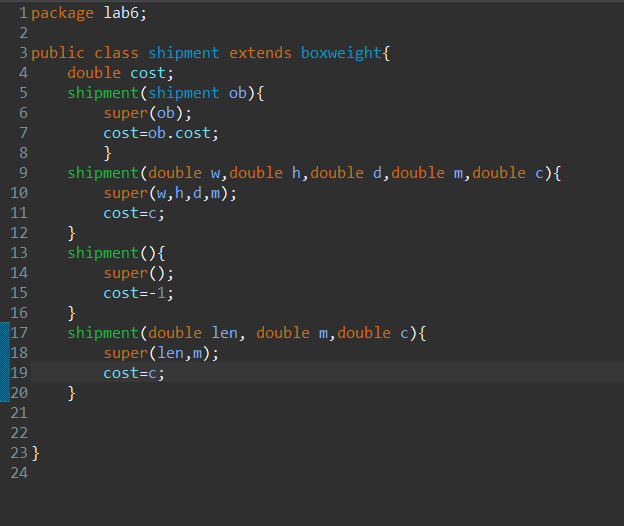
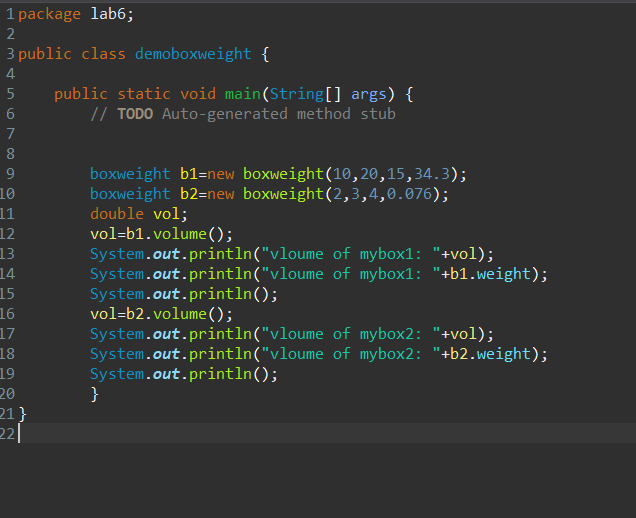
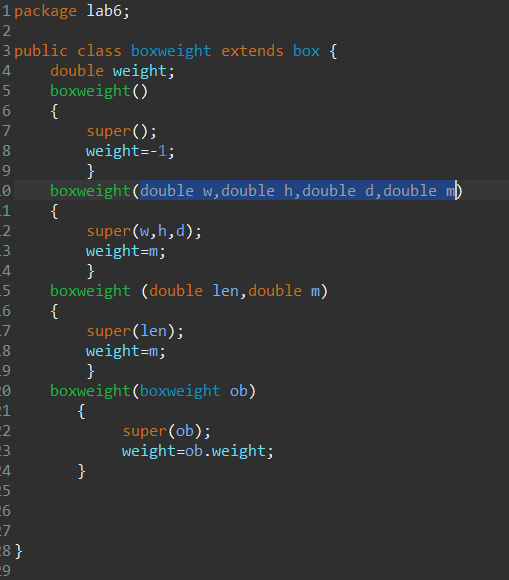
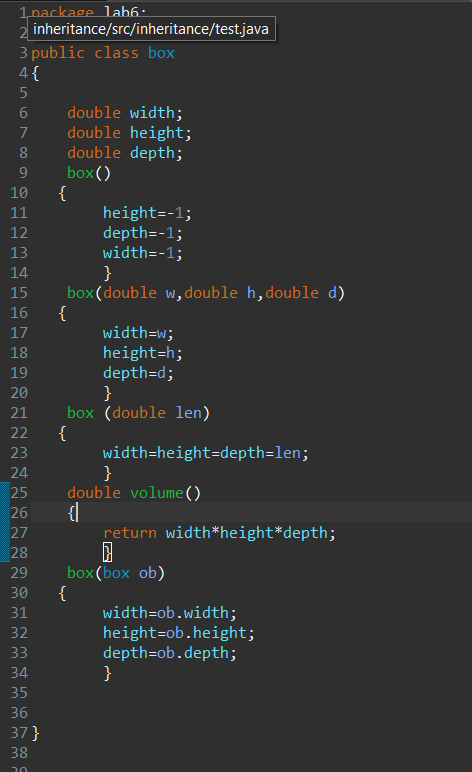
System.out.println("Weight of shipment2 is " + shipment2.weight);

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

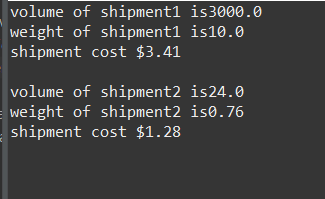
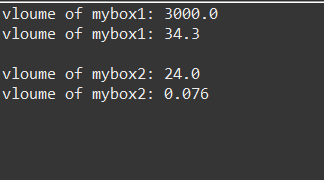
}

}

**Code**



**Output**



**Lab Assignment:**

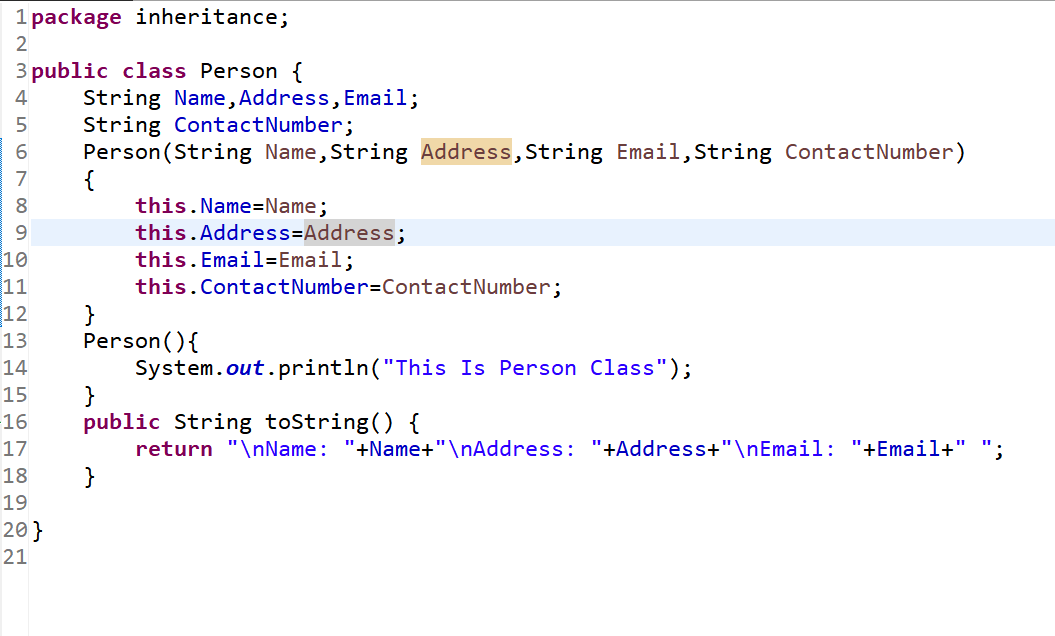
1. **Design a class named Person and its two subclasses named Student and Employee. Make Faculty and Staff subclasses of Employee.**

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

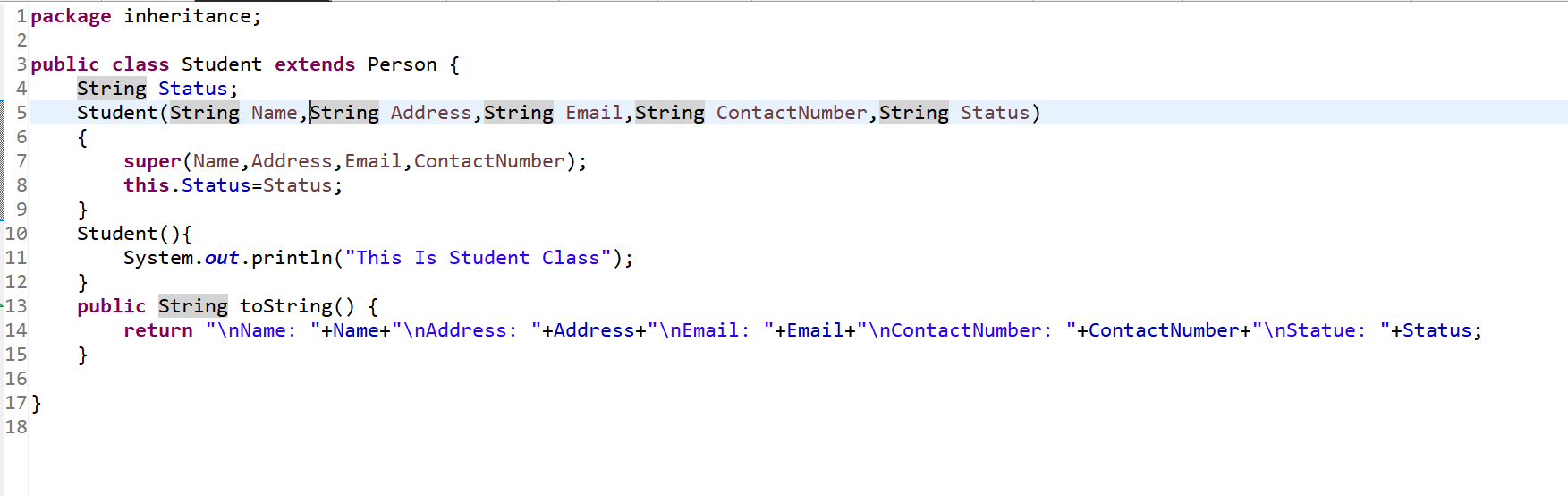
**A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date hired. A faculty member has office hours and a rank. A staff member has a title. Override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

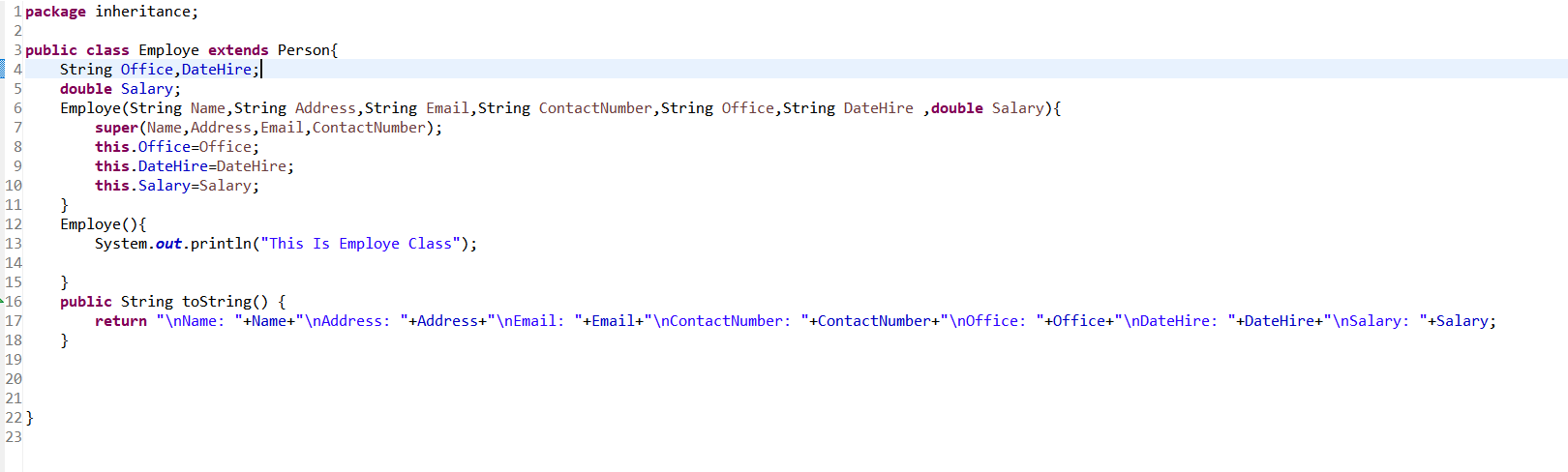
1. **Design a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their toString() methods.**

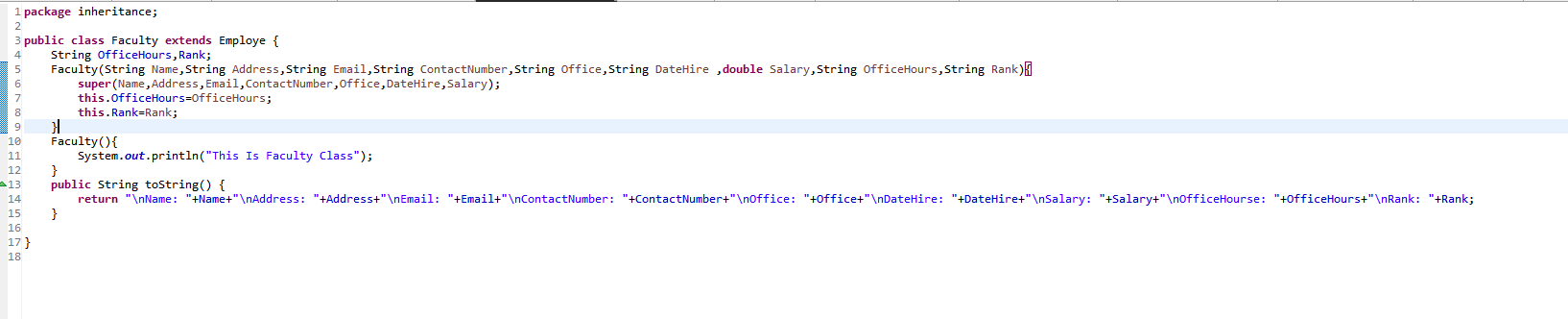
**Code**

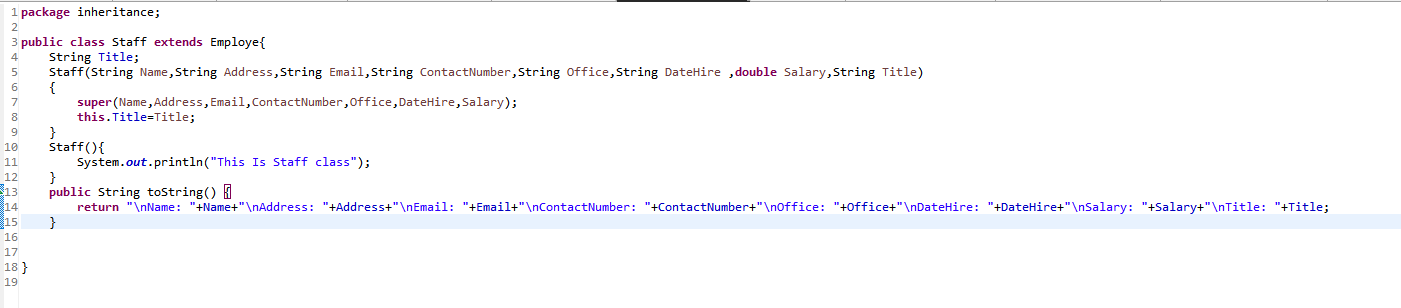


Gig



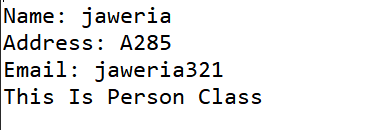


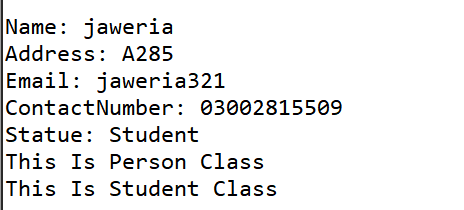


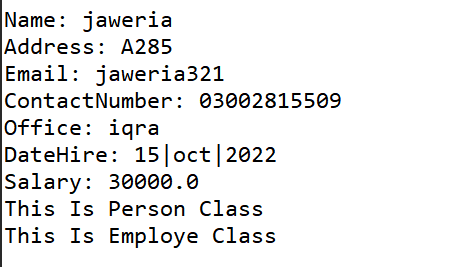


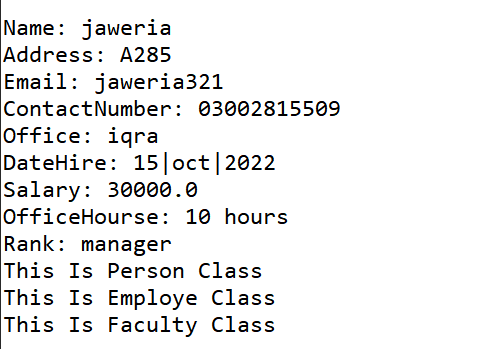


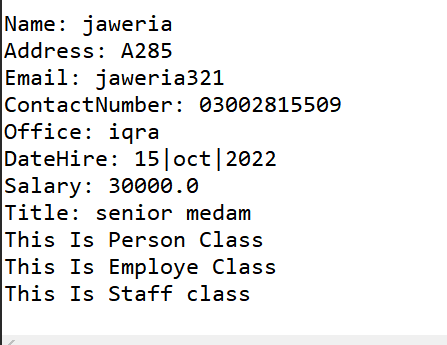
**Output**











}

**Conclusion:**

In this lab we learn abouthow to use inheritance in java programming language

**Lab-07**

**Use of abstract & final**

**Objectives:**

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 {

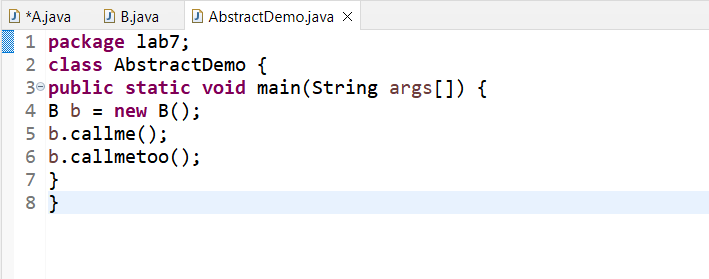
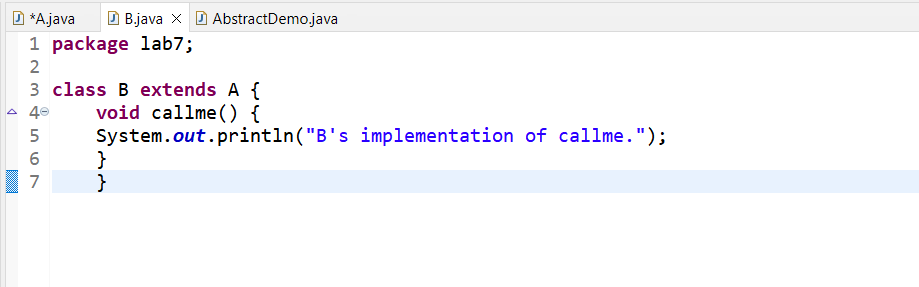
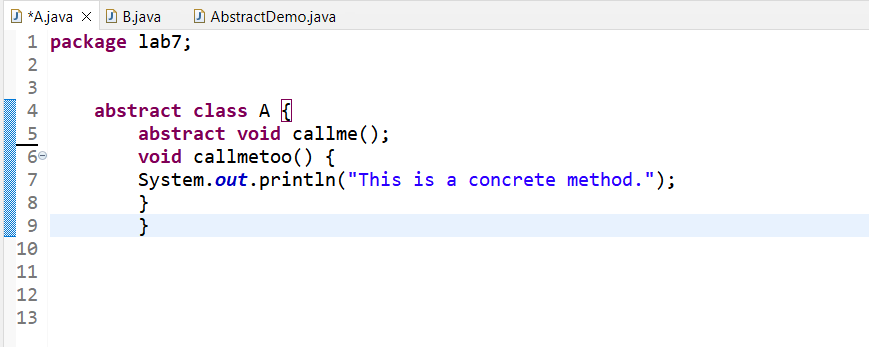
void meth() { // ERROR! Can't override.

System.out.println("Illegal!");

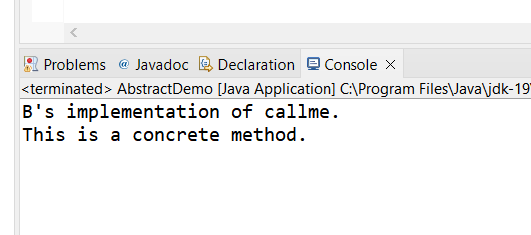
}

}

**Code**



**Ouput**



**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* booleanwithdraw(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.

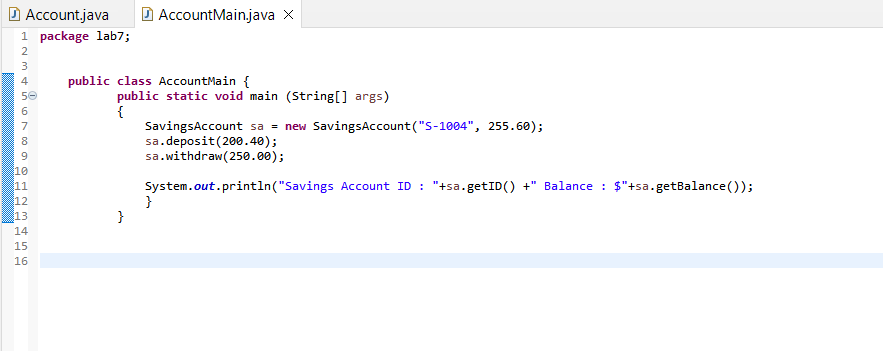
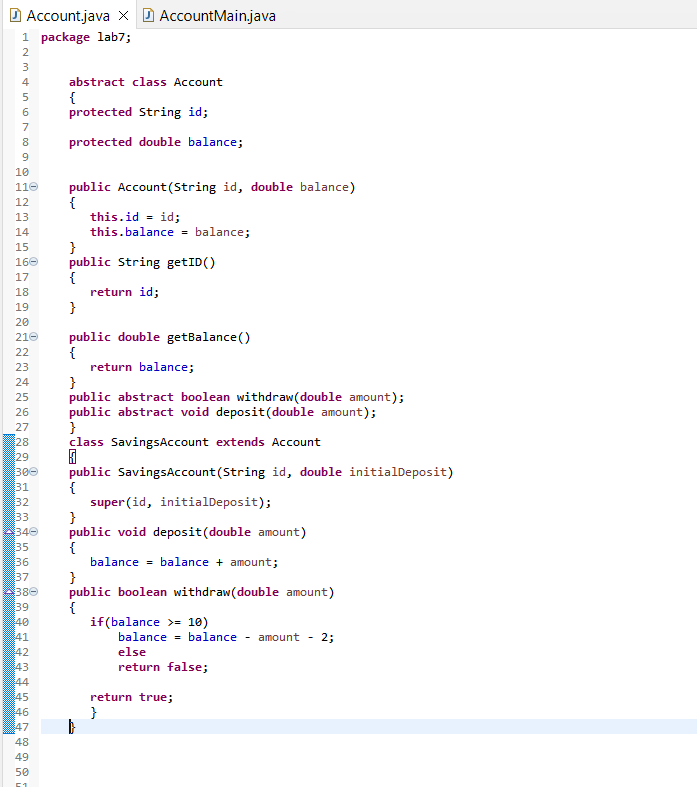
* List of methods:

public void deposit(double amount): // The provided amount is added to the account

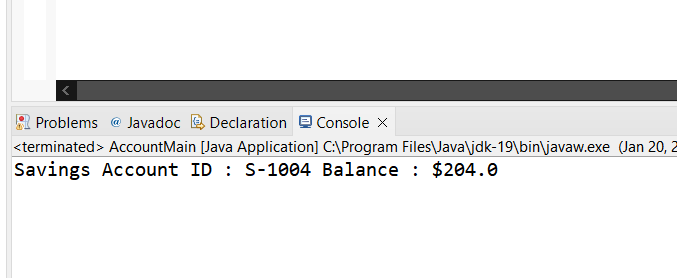
publicbooleanwithdraw(double amount):

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.

**Code**



**Ouput**



**Conclusion:**

In this lab we learn abouthow to use abstract and final in java programming language

**Lab - 08**

**Packages & Interfaces**

**Objectives:**

**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:

packagepkg;

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(inti=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:

Java MyPack.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:

Java AccountBalance

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);

}

}

**interface Animal**

**{**

**public void move();**

**}**

**interface Predator**

**{**

**public void hunt();**

**}**

**class Wolf implements Animal, Predator**

**{**

**private int length;**

**public void move()**

**{**

**System.out.println("Wolf is moving!");**

**}**

**public void hunt()**

**{**

**System.out.println("Wolf is hunting!");**

**}**

**}**

**class Fox implements Animal, Predator**

**{**

**private String fur;**

**public void move()**

**{**

**System.out.println("Fox is moving!");**

**}**

**public void hunt()**

**{**

**System.out.println("Fox is hunting!");**

**}**

**}**

**import java.util.ArrayList;**

**class AnimalPredators**

**{**

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

**ArrayList<Animal> animals=new ArrayList<Animal>();**

**animals.add(new Wolf());**

**animals.add(new Fox());**

**animals.add(new Fox());**

**for(Animal animal: animals)**

**animal.move();**

**ArrayList<Predator> predators=new ArrayList<Predator>();**

**predators.add(new Wolf());**

**predators.add(new Fox());**

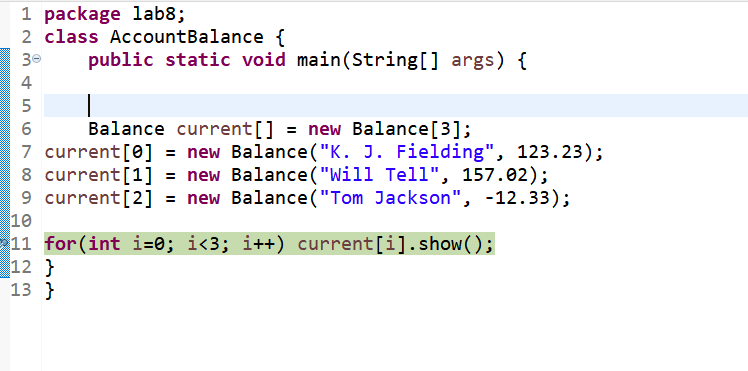
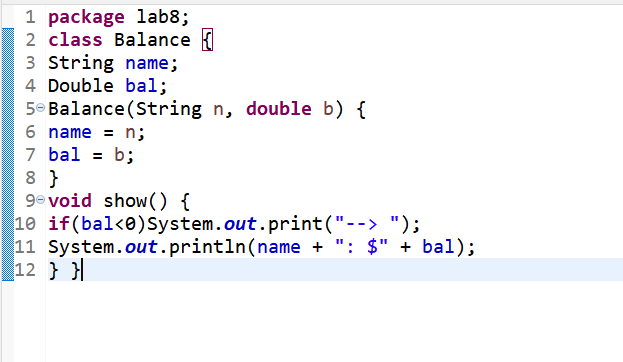
**predators.add(new Fox());**

**for(int i=0; i<predators.size(); i++)**

**predators.get(i).hunt();**

**}**

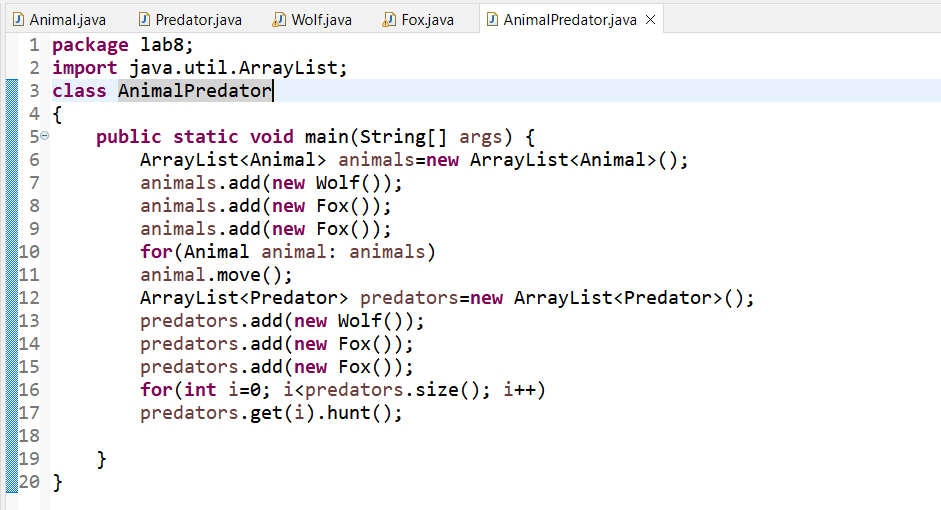
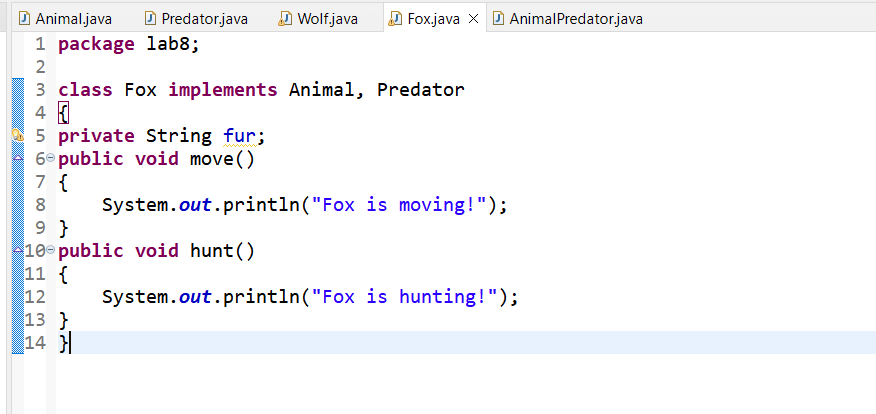
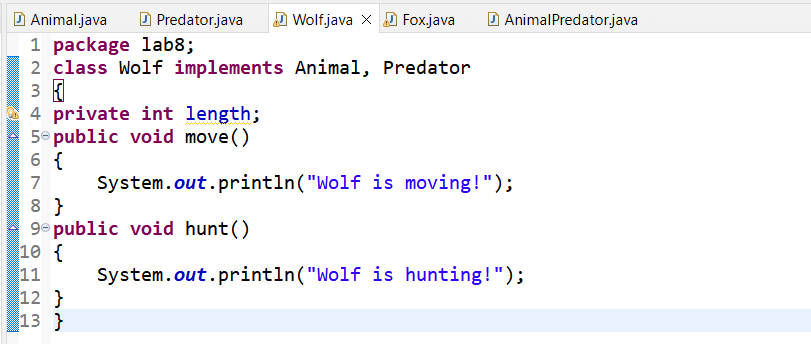
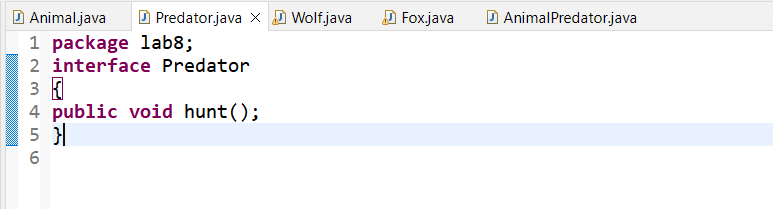
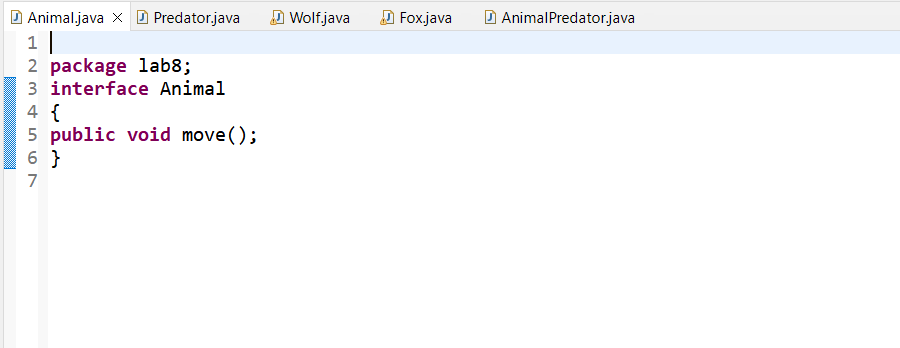
**Code**



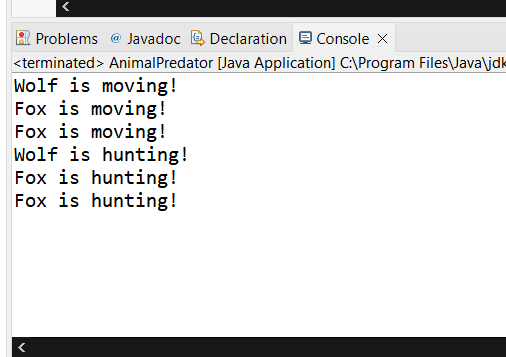
**Lab Assignment:**

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

**Code**



**Output**



1. **Design 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.**

**Code**

**package Lab8;**

**public class CharSequenceMain implements CharSequence {**

**private String s;**

**public CharSequenceMain(String s) {**

**this.s = s;**

**}**

**private int fromEnd(int i) {**

**return s.length() - 1 - i;**

**}**

**public char charAt(int i) {**

**if ((i < 0) || (i >= s.length())) {**

**throw new StringIndexOutOfBoundsException(i);**

**}**

**return s.charAt(fromEnd(i));**

**}**

**public int length() {**

**return s.length();**

**}**

**public CharSequence subSequence(int start, int end) {**

**if (start < 0) {**

**throw new StringIndexOutOfBoundsException(start);**

**}**

**if (end > s.length()) {**

**throw new StringIndexOutOfBoundsException(end);**

**}**

**if (start > end) {**

**throw new StringIndexOutOfBoundsException(start - end);**

**}**

**StringBuilder sub =**

**new StringBuilder(s.subSequence(fromEnd(end), fromEnd(start)));**

**return sub.reverse();**

**}**

**public String toString() {**

**StringBuilder s = new StringBuilder(this.s);**

**return s.reverse().toString();**

**}**

**private static int random(int max) {**

**return (int) Math.round(Math.random() \* (max+1));**

**}**

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

**CharSequenceMain s =**

**new CharSequenceMain("Write a class that implements the CharSequence interface found in the java.lang package.");**

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

**System.out.print(s.charAt(i));**

**}**

**System.out.println("");**

**int start = random(s.length() - 1);**

**int end = random(s.length() - 1 - start) + start;**

**System.out.println(s.subSequence(start, end));**

**System.out.println(s);**

**}**

**}**

**Conclusion:**

In this lab we learn abouthow to use package and interface in java programming language

**Lab-09**

**Introducing JavaFx– Java GUI**

**Objectives:**

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:**

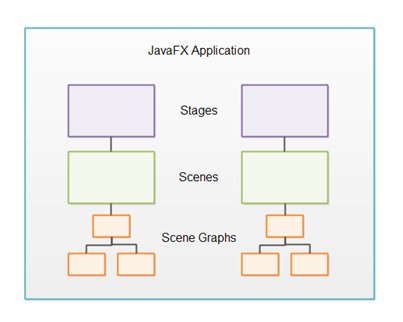
A graphical user interface (GUI) presents a user friendly mechanism for interacting with an application. GUIs are built from GUI components. These are sometimes called controls or widgets. A GUI component is an object with which the user interacts via mouse, the keyboard, or another form of input.

**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.

**JavaFxApplication 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.

**JavaFxControls**

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

Application.launch(args);

}

public void start(Stage stage) throws Exception {

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

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

//heading.setFont(Font);

//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();

}

}

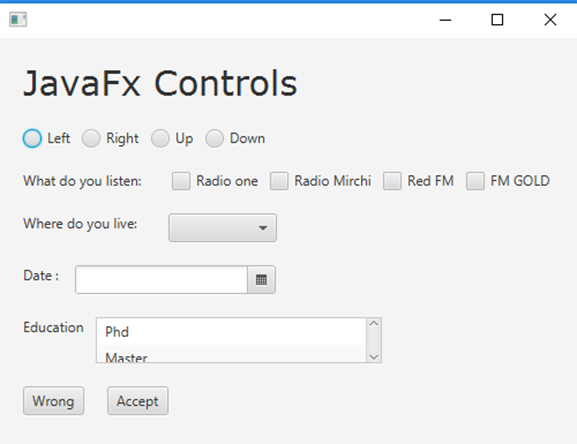


Figure 9.1: JavaFx Controls

**Login window:**  
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.

import javafx.application.Application;

import javafx.event.\*;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.effect.\*;

import javafx.scene.layout.\*;

import javafx.scene.paint.Color;

import javafx.scene.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) {

Application.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();

}

}

**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.

importjavafx.application.Application;

importjavafx.application.Platform;

importjavafx.scene.Scene;

importjavafx.scene.control.\*;

importjavafx.scene.layout.BorderPane;

importjavafx.scene.paint.Color;

importjavafx.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");

CheckMenuItemhtml MenuItem = new CheckMenuItem("HTML");

htmlMenuItem.setSelected(true);

webMenu.getItems().add(htmlMenuItem);

CheckMenuItemcssMenuItem = new CheckMenuItem("CSS");

cssMenuItem.setSelected(true);

webMenu.getItems().add(cssMenuItem);

Menu sqlMenu = new Menu("SQL");

ToggleGrouptGroup = 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 tutorialManeu = new Menu("Tutorial");

tutorialManeu.getItems().addAll(

new CheckMenuItem("Java"),

new CheckMenuItem("JavaFX"),

new CheckMenuItem("Swing"));

sqlMenu.getItems().add(tutorialManeu);

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

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Conclusion:**

In this lab we learn abouthow to use java fx in java programming language

**Lab-10**

**Exploring JavaFx**

**Objectives:**

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.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 {

primaryStage.setTitle("FlowPane Experimwnt");

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);

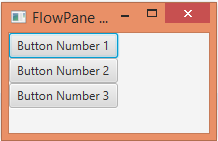
primaryStage.show();

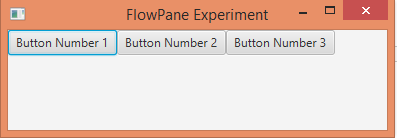
}

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();

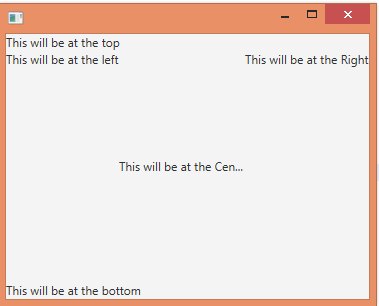
    }

    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);

gridPane.add(button2, 1, 0);

gridPane.add(button3, 2, 0);

gridPane.add(button4, 0, 1);

gridPane.add(button5, 1, 1);

gridPane.add(button6, 2, 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(500);

primaryStage.show();

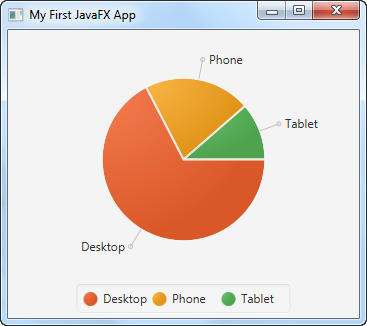
}

public static void main(String[] args) {

Application.launch(args);

}

}



**Conclusion:**

In this lab we learn abouthow to use java fx in java programming language

**LAB - 11**

**JavaFxApplication**

**Objectives:**

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 173.1: 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 extends Application{

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();

}

});

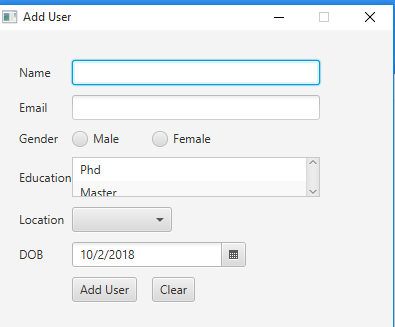
scene= new Scene(layout);

}

public static void main(String[] args) {

launch(args);

}

 @Override

public void start(Stage primaryStage) {

}

}

**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.

importjavafx.application.Application;

importjavafx.application.Platform;

importjavafx.scene.Scene;

importjavafx.scene.control.CheckMenuItem;

importjavafx.scene.control.Menu;

importjavafx.scene.control.MenuBar;

importjavafx.scene.control.MenuItem;

importjavafx.scene.control.RadioMenuItem;

importjavafx.scene.control.SeparatorMenuItem;

importjavafx.scene.control.ToggleGroup;

importjavafx.scene.layout.BorderPane;

importjavafx.scene.paint.Color;

importjavafx.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");

CheckMenuItemhtml MenuItem = new CheckMenuItem("HTML");

htmlMenuItem.setSelected(true);

webMenu.getItems().add(htmlMenuItem);

CheckMenuItemcssMenuItem = new CheckMenuItem("CSS");

cssMenuItem.setSelected(true);

webMenu.getItems().add(cssMenuItem);

Menu sqlMenu = new Menu("SQL");

ToggleGrouptGroup = 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 tutorialManeu = new Menu("Tutorial");

tutorialManeu.getItems().addAll(

new CheckMenuItem("Java"),

new CheckMenuItem("JavaFX"),

new CheckMenuItem("Swing"));

sqlMenu.getItems().add(tutorialManeu);

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

primaryStage.setScene(scene);

primaryStage.show();

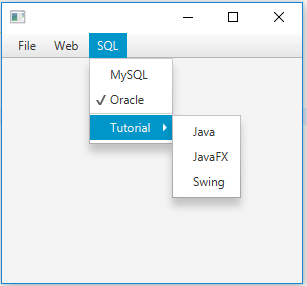
}

public static void main(String[] args) {

launch(args);

}

}



**Conclusion:**

In this lecture we learnt how to use java fx Application in java programming language

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**