

LAB MANUAL

Course: CSC241-Object Oriented Programming



Department of Computer Science

COMSATS UNIVERSITY ISLAMABAD
Lahore Campus

Table of Contents

Lab #	Topics Covered	Page #
Lab # 01	Basic Syntax of Java,	03
Lab # 02	Difference between Object Oriented and Procedural Programming	11
Lab # 03	Defining classes, Objects Constructors in JAVA	20
Lab # 04	Controlling access to class members – Encapsulation	29
Lab # 05	Static Data members and Methods	37
Lab # 06	User Input in Java	47
Lab # 07	Enum	54
Lab # 08	Composition / Containership(Has-a relationship)	58
Lab # 09	Arrays in Java (1D Array)	67
Lab # 10	Arrays in Java (Multidimensional Array)	73
Lab # 11	Lab Sessional 1	
Lab # 12		
Lab # 13	Array List in Java	79
Lab # 14	Inheritance	84
Lab # 15	Inheritance	84
Lab # 16	Method Overriding	93
Lab # 17	Polymorphism	100
Lab # 18	Polymorphism	
Lab # 19	Abstraction	107
Lab # 20	Abstraction	112
Lab # 21	Interfaces	
Lab # 22	Lab Sessional 2	
Lab # 23		
Lab # 24	Exception Handling	123
Lab # 25	Exception Handling	
Lab # 26	Exception Handling	
Lab # 27	File Handling	132
Lab # 28	File Handling	
Lab # 29	Terminal Examination	
Lab # 30		

Statement Purpose:

Objective of this lab is to make students understand the basic syntax of java specifically conditions and loops

Activity Outcomes:

The students will learn the basic syntax of java.

Instructor Note:

The Students should have knowledge about structured programming.

1. Stage J (Journey)

Introduction

Microprocessor are key component in consumer-electronics 1991 - Sun Microsystems funded an internal project led by James Gosling, to write a language that shall execute on all microprocessor for consumer-electronics. In 1993, the web wide adaptation given rise to Java because Sun Microsystems launched features to create web applications. Java become popular because of web, not for its initial goal. In 2009, Oracle acquired Sun Microsystems.

Some features of Java are given below:

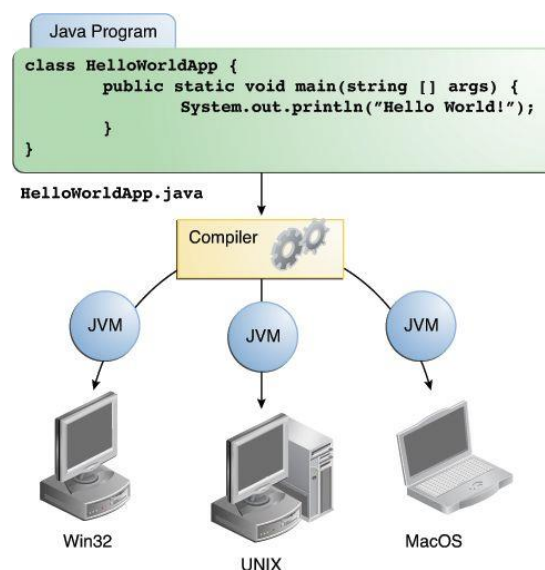
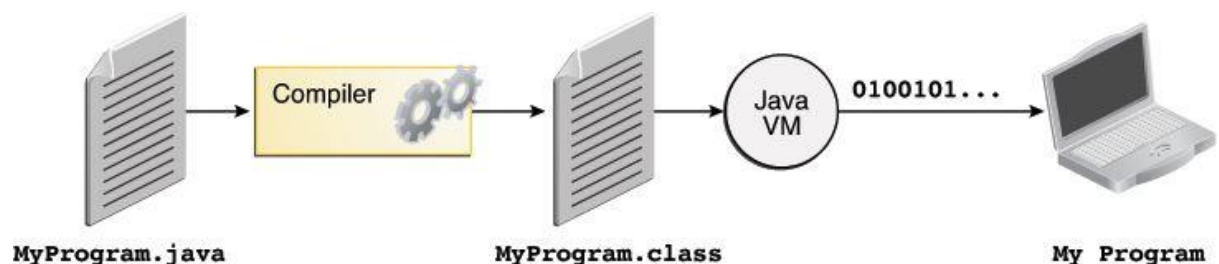
Portability: Write Once, Run Anywhere

Robust, Secure and Reliable – No pointer arithmetic, compile time error checking, bytecode verification, etc.

Strong libraries of components available

Supports distributed applications, multithreading

Life cycle of java program is given below

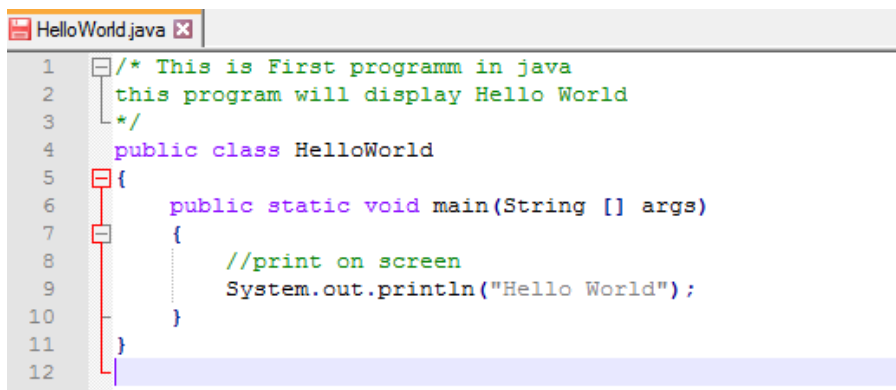


2. Stage a1 (apply)

Lab Activities

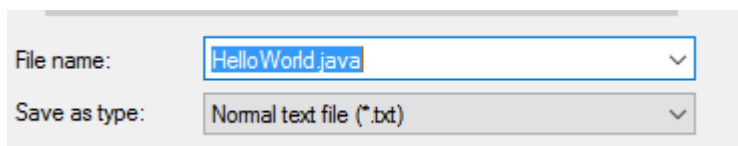
Activity 1

This example will demonstrate use of notepad++ for executing simple java program. Write code in notepad or notepad++. Write the code which displays “Hello World”.



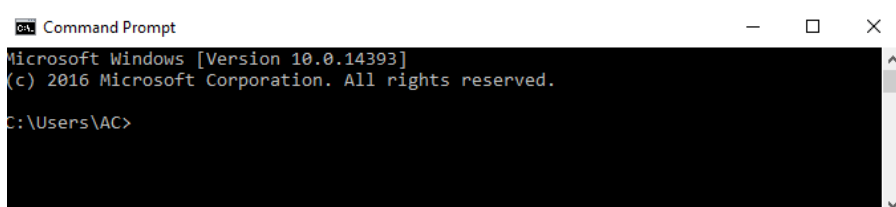
```
1  /* This is First programm in java
2  this program will display Hello World
3  */
4  public class HelloWorld
5  {
6      public static void main(String [] args)
7      {
8          //print on screen
9          System.out.println("Hello World");
10     }
11 }
12
```

Click on file and then click on save as. File name should be same as class name.



Create a folder in C drive. Save the program in new created folder.

Now open the command prompt from All Programs -> Accessories -> command prompt. Following window will be opened.



Write following commands on command prompt.

cd\ (change directory)

cd FolderName

To Compile:

javac HelloWorld.java

'javac' is not recognized as an internal or external command,
operable program or batch file.

C -> program files -> java -> jdk version -> bin -> right click -> properties -> copy path

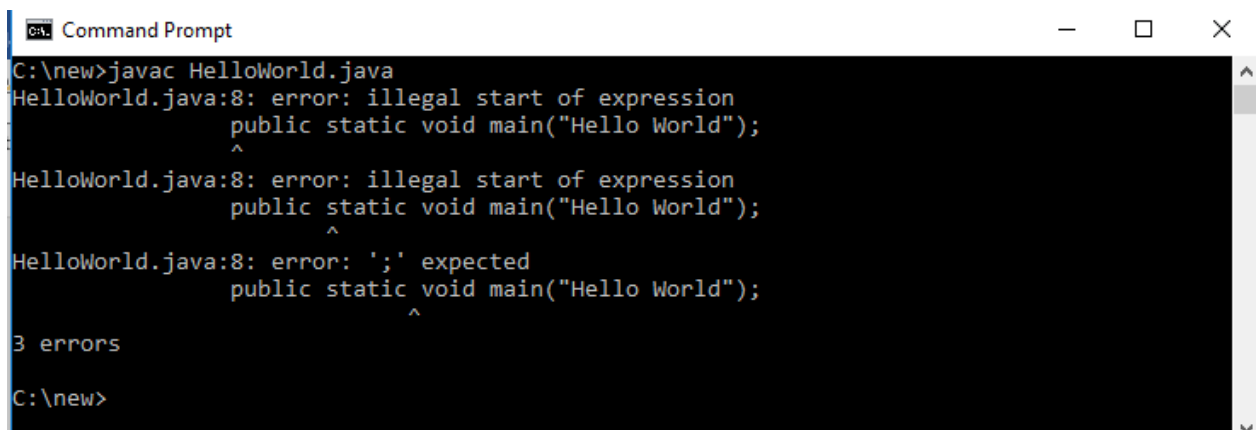
C:\Program Files\Java\jdk1.8.0_161\bin

TEMPORARY PATH SETTING

Command prompt -> set path= C:\Program Files\Java\jdk1.8.0_161\bin

Javac HelloWorld.java

If there is no syntax error, then file will be compiled successfully and a new file with same name will be created in same directory with extension “.class”. You can verify it by checking your folder yourself.

A screenshot of a Windows Command Prompt window titled "C:\> Command Prompt". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the following text:

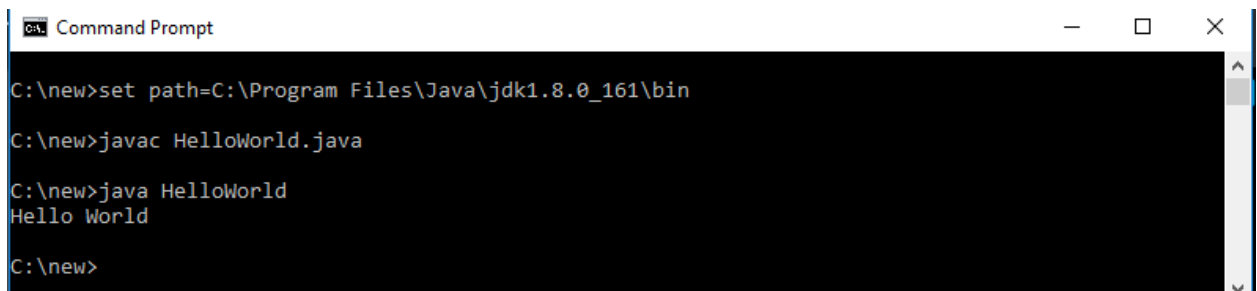
```
C:\new>javac HelloWorld.java
HelloWorld.java:8: error: illegal start of expression
        public static void main("Hello World");
        ^
HelloWorld.java:8: error: illegal start of expression
        public static void main("Hello World");
        ^
HelloWorld.java:8: error: ';' expected
        public static void main("Hello World");
        ^
3 errors
C:\new>
```

Write now it is showing errors. I will correct the error and recompile the file after saving it. Error is rectified. In addition, compiler has not shown any error.

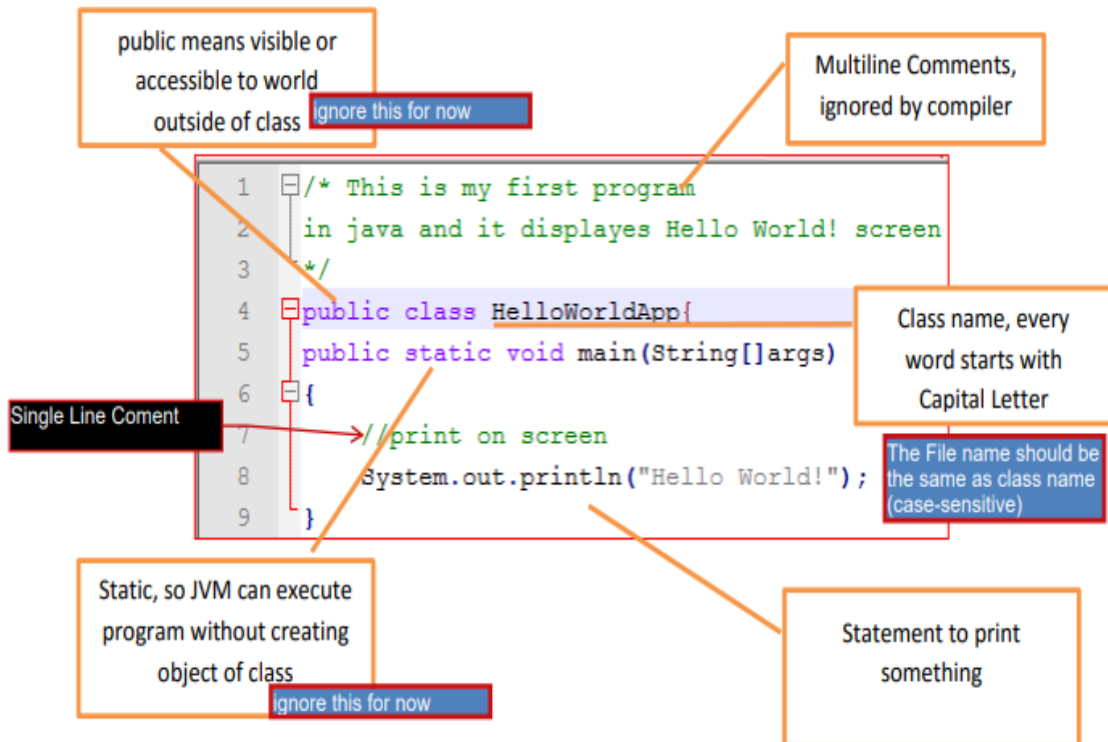
To Run:

java HelloWorld

Now write java FileName and press enter, if there is no logical error then you will see the output “HelloWorld!” at screen.

A screenshot of a Windows Command Prompt window. The title bar reads "C:\ Command Prompt". The window has standard minimize, maximize, and close buttons. The command history is as follows:
C:\new>set path=C:\Program Files\Java\jdk1.8.0_161\bin
C:\new>javac HelloWorld.java
C:\new>java HelloWorld
Hello World
C:\new>
The text is displayed in a monospaced font on a black background with a vertical scrollbar on the right.

Explanation of Program

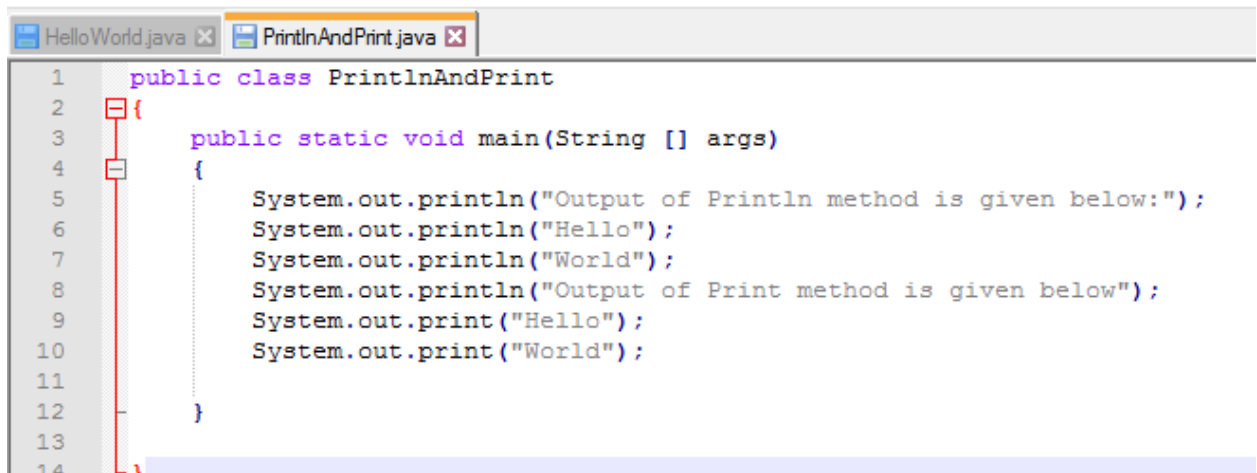


Common Errors

1. If name of file and name of class is different
2. If main method parenthesis is empty or wrong parameter are given.
3. If any matching bracket is missing
4. Semicolon is missing
5. If main method is private and not public
6. If class is private and not public
7. If main method is not static

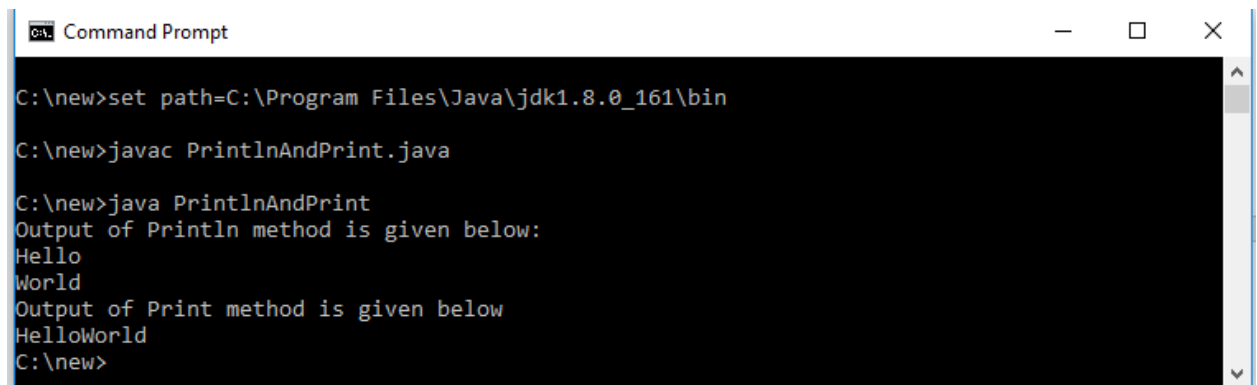
Print vs Println

The **print**("aString") method **prints** just the string "aString", but does not move the cursor to a new line. Hence, subsequent **printing** instructions will **print** on the same line. The **println**("aString") method **prints** the string "aString" and moves the cursor to a new line.



```
1 public class PrintlnAndPrint
2 {
3     public static void main(String [] args)
4     {
5         System.out.println("Output of Println method is given below:");
6         System.out.println("Hello");
7         System.out.println("World");
8         System.out.println("Output of Print method is given below");
9         System.out.print("Hello");
10        System.out.print("World");
11    }
12 }
13
14
```

Output:



```
Command Prompt

C:\new>set path=C:\Program Files\Java\jdk1.8.0_161\bin
C:\new>javac PrintlnAndPrint.java
C:\new>java PrintlnAndPrint
Output of Println method is given below:
Hello
World
Output of Print method is given below
HelloWorld
C:\new>
```

3. Stage ▼ (verify)

Home Activities

Activity 1

Write java code that prints table of 5.

Activity 2

Write java code that prints even and odd numbers from 1 to 10.

Activity 3

Write a code, to check whether a given number is positive number or negative number.

Activity 4

Write java code, which prints sum of first ten natural numbers.

Activity 5

Write java code, which prints factorial of given number.

4. Stage a2 (access)

Assignment

Task 1: Write java code, which prints grades of students according to marks obtained.

If marks are greater than or equal to 90 print A grade

If marks are greater than or equal to 80 and less than 90 print B grade

If marks are greater than or equal to 70 and less than 80 print C grade

If marks are greater than or equal to 60 and less than 70 print D grade

If marks are greater than or equal to 50 and less than 60 print E grade

If marks are less than 50 print F grade.

Task 2: (The number-word program) Write a program, which inputs a one-digit number from the user (i.e. 0-9). The program should then print that number in words, e.g. “Zero” for 0, “One” for 1, “Two” for 2, and so on. If the user does not enter a one-digit number, then program should display an error: “Invalid number!”

Task 3: Write java code, which prints the number in reverse order.

Expected Output: Number 1234

Number in reverse order: 4321

Statement Purpose:

Objective of this lab is to make students understand the difference between object oriented and procedural approaches to programming

Activity Outcomes:

The student will understand the advantages of using OOP

The student will understand the difference between procedural and object oriented approaches

The student will be able to understand difference between class and object.

Instructor Note:

The Students should have knowledge about structured programming.

The students should brainstorm about the scenarios given in activities; in order to model them in terms of Objects.

1. Stage J (Journey)

Introduction

Procedural programming uses a list of instructions to tell the computer what to do step-by-step. Procedural programming relies on procedures, also known as routines or subroutines. A procedure contains a series of computational steps to be carried out.

Procedural programming is intuitive in the sense that it is very similar to how you would expect a program to work. If you want a computer to do something, you should provide step-by-step instructions on how to do it. It is, therefore, no surprise that most of the early programming languages are all procedural. Examples of procedural languages include Fortran, COBOL and C, which have been around since the 1960s and 70s.

Object-oriented programming, or OOP, is an approach to problem-solving where all computations are carried out using objects. An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You would also expect a person to be able to do something, such as walking. This would be considered a method of the person.

A method in object-oriented programming is like a procedure in procedural programming. The key difference here is that the method is part of an object. In object-oriented programming, you organize your code by creating objects, and then you can give those objects properties and you can make them do certain things.

One of the most important characteristics of procedural programming is that it relies on procedures that operate on data - these are two separate concepts. In object-oriented programming, these two concepts are bundled into objects. This makes it possible to create more complicated behavior with less code. The use of objects also makes it possible to reuse code. Once you have created an object with more complex behavior, you can use it anywhere in your code.

The world around us is made up of objects, such as people, automobiles, buildings, streets, and so forth. Each of these objects has the ability to perform certain actions, and each of these actions has some effect on some of the other objects in the world.

OOP is a programming methodology that views a program as similarly consisting of objects that interact with each other by means of actions.

Object-oriented programming has its own specialized terminology. The objects are called, appropriately enough, objects. The actions that an object can take are called methods. Objects of the same kind are said to have the same type or, more often, are said to be in the same class.

For example, in an airport simulation program, all the simulated airplanes might belong to the same class, probably called the Airplane class. All objects within a class have the same methods. Thus, in a simulation program, all airplanes have the same methods (or possible actions), such as taking off, flying to a specific location, landing, and so forth. However, all simulated airplanes are not identical. They can have different characteristics, which are indicated in the program by associating different data (that is, some different information) with each particular airplane object. For example, the data associated with an airplane object might be two numbers for its speed and altitude.

Things that are called procedures, methods, functions, or subprograms in other languages are all called methods in Java. In Java, all methods (and for that matter, any programming constructs whatsoever) are part of a class.

2. Stage **a1** (apply)

Lab Activities

Activity 1

The example demonstrates the difference in approach if we want to find the circumference of circle.

Solution:

Procedural Approach	Object Oriented Approach
<pre> Public class Circle{ int radius; Public void setRadius(int r) { radius = r;} Public void showCircumference() { double c = 2*3.14*radius; System.out.println("Circumferenceis"+ c); } Public static void main() { setRadius(5); showCircumference(); //output would be 31.4 setRadius(10); showCircumference(); // output would be 62.8 } </pre>	<pre> Public class Circle{ Private int radius; Public void setRadius(int r) { radius = r;} Public void showCircumference() { double c = 2*3.14* radius; System.out.println("Circumference is"+ c); } } Public class runner { Public static void main() { Circle c1= new circle(); c1.setRadius(5); c1.showCircumference(); } } </pre>

}	//output would be 31.4; it belongs to c1 Circle c2= new circle(); c2.setRadius(10); c2.showCircumference(); //output would be 62.8; it belongs to c2 } }
---	--

Activity 2

Activity 1:

Consider the concept of a CourseResult. The CourseResult should have data members like the student name, course name and grade obtained in that course. This concept can be represented in a class as follows:

Solution:

```
Public class CourseResult
{
    Public String studentname;
    Public String coursename;
    Public String grade;
    public void display()
    {
        System.out.println("Student Name is: " + studentname + "Course Name is: " +
        coursename +
        "Grade is: " + grade);
    }
}
```

```
Public class CourseResultRun
{
    public static void main(String[]args)
    {
        CourseResult c1=new CourseResult ();
        c1.studentName= "Ali";
        c1.courseName= "OOP";
        c1.grade= "A";
        c1.display();

        CourseResult c2=new CourseResult ();
```

```

        c2.studentName= "Saba";
        c2.courseName= "ICP";
        c2.grade= "A+";
        c2.display();
    }
}

```

Note that both objects; c1 and c2 have three data members, but each object has different values for their data members.

Activity 3:

The example below represents a Date class. As date is composed of three attributes, namely month, year and day; so the class contains three Data Members. Now every date object will have these three attributes, but each object can have different values for these three

Solution:

```

public class Date
{
    public String month;
    public int day;
    public int year; //a four digit number.

    public void displayDate()
    {
        System.out.println(month + " " + day + ", " + year);
    }
}

public static void main(String[] args)
{
    Date date1, date2;

    date1 = new Date();

    date1.month = "December";

    date1.day = 31;

    date1.year = 2012;

    System.out.println("date1:");

    date1.display();
}

```

```
date2 = new Date();  
date2.month = "July";  
date2.day = 4;  
date2.year = 1776;  
System.out.println("date2:");  
date2.display();  
}  
}
```

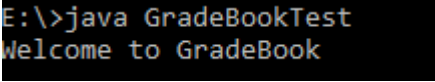
3. Stage v (verify)

Home Activities

Activity 1:

Write java code to create the *GradeBook* class that contains a *displayMessage* method to displays a message on the screen. You will need to make an object of this class and call its method to execute display this message “Welcome to GradeBook” Now declare a separate class that contains a main method. The *GradeBookTest* class declaration will contain the main method that will control your application’s execution.

Expected Output:



```
E:\>java GradeBookTest  
Welcome to GradeBook
```

Activity 2:

A sample Dog class. Attributes are name and breed. Methods are speak, sleep and eat.


```

1 public class Dog { //class starts
2     //-----Attributes
3     public String name = "max";
4     public String breed = "pet";
10    //-----Methods
11    public void speak()
12    {
13        System.out.println("woof");
14    }
15    public void sleep()
16    {
17        System.out.println("sleeping");
18    }
19    public void eat()
20    {
21        System.out.println("eating");
22    }
23 } //class ends

```

Experiment:

1. Compile the above class (without main) using `javac filename.java`. Does it compile? Why?
2. If above experiment is successful, run the compiled .class file. What happens? Why?

Program

A sample DogTest class (in separate file)

```

1 public class DogTest {
2
3     public static void main(String[] args) {
4         Dog d1=new Dog();
5         System.out.printf("My dog %s is ",d1.name);
6         d1.sleep();
7         System.out.printf("Hello %s \n",d1.name);
8         d1.speak();
9
10    }
11 }

```

Expected Output:

```

General Output
-----Con
-----
My dog max is sleeping
Hello max
woof

```

Activity 3:

Write a class Circle, which will model the functionality of a Circle.

1. Attributes
 - ▶ radius
2. Methods
 - ▶ *calculateArea()*: To compute area
 - ▶ *calculatePerimeter()*: To compute perimeter

Note: For value of pi, use Math.PI

Expected Output:

```
Radius is 3.50
Area is: 38.47
Perimeter is: 21.98
```

4. Stage a2 (access)

Assignment

Task 1: Write a class Rectangle, which will model the functionality of a Rectangle.

1. Attributes
 - ▶ Length
 - ▶ Width
2. Methods
 - ▶ *calculateArea()*: To compute area
 - ▶ *calculatePerimeter()*: To compute perimeter

Expected Output:

```
Length is 4
Width is 2
Area is: 8.00
Perimeter is: 12.00
```

Task 2: Write a class Account, which will model the functionality of a bank account.

1. Attributes
 - ▶ AccountTitle
 - ▶ TotalBalance
2. Methods
 - ▶ *deposit(double amountToDeposit)*: To deposit amount to account

- ▶ *withdraw(double amountToWithdraw)*: To withdraw amount from account. In this method, you have check withdrawal amount should be less than total amount.

Statement Purpose:

Objective of this lab is to understand the importance of classes and construction of objects using constructors.

Activity Outcomes:

The student will be able to declare a classes and objects.

The student will be able to declare member functions and member variables of a class.

The student will be able to declare overloaded constructors.

Instructor Note:

The student should have understanding objects, classes and methods.

1. Stage J (Journey)

Introduction

- **Data Abstraction**

Abstraction is the process of recognizing and focusing on important characteristics of a situation or object and leaving/filtering out the un-wanted characteristics of that situation or object. For example a person will be viewed differently by a doctor and an employer.

A doctor sees the person as patient. Thus he is interested in name, height, weight, age, blood group, previous or existing diseases etc of a person.

An employer sees a person as an employee. Therefore, employer is interested in name, age, health, degree of study, work experience etc of a person.

- **Class and Object:**

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an

object. A class serves as a plan, or blueprint. It specifies what data and what functions **will be included in objects of that class. An object is often called an “instance” of a**

class.

- **Instance Variables and Methods:**

Instance variables represent the characteristics of the object and methods represent the behavior of the object. For example length & width are the instance variables of class Rectangle and Calculatearea() is a method.

Instance variables and methods belong to some class, and are defined inside the class to which they belong.

Syntax:

```

public class Class_Name
{
    Instance_Variable_Declaration_1
    Instance_Variable_Declaration_2
    ...
    Instance_Variable_Declaration_Last

    Method_Definition_1
    Method_Definition_2
    ...
    Method_Definition_Last
}

```

- **Constructors:**

It is a special function that is automatically executed when an object of that class is created. It has no return type and has the same name as that of the class. It is normally defined in classes to initialize data members. A constructor with no parameters is called a no-argument constructor. A constructor may contain arguments which can be used for initiation of data members.

Syntax:

```

class_name( )
{
    Public class_name()
    {
        //body
    }
}

```

```

Public class_name(type var1, type var2)
{
    //body
}
}

```

If your class definition contains absolutely no constructor definitions, then Java will automatically create a no-argument constructor. If your class definition contains one or more constructor definitions, then Java does not automatically generate any constructor; in this case, what you define is what you get. Most of the classes you define should include a definition of a no-argument constructor.

Constructor chaining is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

2. Stage a1 (apply)

Lab Activities

Activity 1

The following example demonstrates the use of constructors

Solution:

```
public class Rectangle
{
    Public int length, width;

    public Rectangle()
    {
        length = 5; width = 2;
    }

    public Rectangle(int l, int w)
    {
        length = l; width = w;
    }

    Public intCalculatearea ()
    {
        return (length*width);
    }
}

Public class runner
{
    Public static void main ()
    {
        Rectangle rect = new Reactangle();

        System.out.println(rect.calculateArea( ));

        Rectangle rect = new Reactangle(10,20);
        System.out.println(rect. calculateArea ( ));
    }
}
```

Activity 2

The following example shows the declaration of class Point. It has two data members that represent the x and y coordinate. Create two constructors and a function to move the point. The runner class will create an object of Point class and move function will be called.

Solution:

```
public class Point{

    private int x;
    private int y;
    public Point(){
        x=1;
        y=2;
    }
    public Point(int a, int b){
        x=a;
        y=b;
    }
    public void setX(int a){
        x=a;
    }
    public void setY(int b){
        y=b;
    }
    public void display(){
        System.out.println("x coordinate = "+x+" y coordinate = "+y);
    }
    public void movePoint(int a , int b){
        x=x+a;
        y=y+b;
        System.out.println("x coordinate after moving = "+x+" y coordinate after moving
        =
        "+y);
    }

}

Public class runner
{
    Public static void main ()
    {
        Point p1 = new Point();
        P1.move(2,3);
        P1.display();
        Point p2 = new Point();
        p2.move(2,3);
        p2.display();
    }
}
```



```
}  
  
}
```

3. Stage v (verify)

Home Activities

Activity 1:

Create a class circle class with radius as data member. Create two constructors (no argument, and two arguments) and a method to calculate Circumference.

Activity 2:

Create a class Account class with balance as data member. Create two constructors (no argument, and two arguments) and methods to withdraw and deposit balance.

Activity 3:

Create a class “Distance” with two constructors (no argument, and two argument), two data members (feet and inches). Also create display function which displays all data members.

Activity 4:

Make a Student class with id, name, age and city. Choose appropriate data types yourself. All attributes should be public.

1. Make an empty constructor and print following statement in constructor
“Constructor with zero argument”

Make a class StudentTest, in main method, instantiate a Student objects and print name and id for of student.

Expected Output:

```
run:  
Constructor with zero argument  
10  
Ali  
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Now create another constructor that takes id and name attributes and assign name and id to instance attribute. In addition, print following statement in constructor.

“Constructor with two parameters”

In main method instantiate, a student object and print name and id of student.

Expected Output:

```
run:
Constructor with zero argument
Constructor with two parameters
10
Ali
```

3. Now create another constructor that takes all attributes and assign name, id, age and city to instance attribute. In addition, print following statement in constructor.

“Constructor with four parameters”

In main method instantiate, another student object and print name and id of student.

Expected Output:

```
run:
Constructor with zero argument
Constructor with two parameters
10
Ali
Constructor with four parameters
20
Maryam
20
Lahore
```

Activity 5

Make a class Employee with following attributes: id, name, salary. Now make following constructors:

1. Constructor that takes no parameter. In this constructor call 2nd constructors
2. Constructor that takes id, name, and print id & name of employee. In this constructor, call 3rd constructor.
3. Fully parameterized constructor and print salary of employee.

Make a class EmployeeTest, in main method, instantiate an employee object (constructor with zero parameter).

Activity 6

Remove the error from following code.

Student.java

```
public class Student{
    int id;
    String name;
    int age;
    String city;
    public Student() {
        System.out.println("This is default constructor");
        this(20,"Maryam");
    }

    public Student(int id, String name) {
        System.out.println("Constructor with two parameters");
        System.out.printf("Id: %d, Nmae: %s\n",id,name);
        this(10,"Ali",20,"Lahore");
    }

    public Student(int id, String name, int age, String city) {
        System.out.println("Constructor with four parameters");
        System.out.printf("Id: %d, Nmae: %s, age: %d, City: %s\n",id,name, age, city);
    }
}
```

StudentTestt.java

```
public class StudentTestt {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Student obj1=new Student();
    }

}
```

4. Stage a2 (access)

Assignment

Task 1:

Write a class Marks with three data members to store three marks. Create two constructors and a method to calculate and return the sum.

Task 2:

Write a class Time with three data members to store hr, min and seconds. Create two constructors and apply checks to set valid time. Also create display function which displays all data members.

Statement Purpose:

The objective of this lab is to teach the students, concept of encapsulation and access modifiers.

Activity Outcomes:

At the end of this lab student will be familiar with the accessing rules of class data members and member functions

Instructor Note:

The Student must understand the procedure for creating a class.

Student should be able to call the methods of a class by using objects.

1. Stage J (Journey)

Introduction

Encapsulation

Information hiding means that you separate the description of how to use a class from the implementation details, such as how the class methods are defined. The programmer who uses the class can consider the implementation details as hidden, since he or she does not need to look at them. Information hiding is a way of avoiding information overloading. It keeps the information needed by a programmer using the class within reasonable bounds. Another term for information hiding is abstraction.

Encapsulation means grouping software into a unit in such a way that it is easy to use because there is a well-defined simple interface. So, encapsulation and information hiding are two sides of the same coin.

Access Modifiers

Java allows you to control access to classes, methods, and fields via access modifiers. The access to classes, constructors, methods and fields are regulated using access modifiers i.e. a class can control what information or data can be accessible by other classes. To take advantage of encapsulation, you should minimize access whenever possible.

Syntax:

```
class class_name {  
    access_specifier type member1;  
    access_specifier type member1;  
    .....  
}
```

The following table describes the access modifiers provided by JAVA.

Modifier	Description
(no modifier)	member is accessible within its package only
Public	member is accessible from any class of any package
Protected	member is accessible in its class package and by its subclasses
Private	member is accessible only from its class

The accessibility rules are depicted in the following table

Protection	Accessed inside Class	Accessed in subclass	Accessed in any Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	yes

Accessors and Mutators

We should always make all instance variables in a class private and should define public methods to provide access to these members. Accessor methods allow you to obtain the data. For example, the method `getMonth()` returns the number of the month. Mutator methods allow you to change the data in a class object.

2. Stage **a1** (apply)

Lab Activities

Activity 1

The following example shows the declaration of class `Circle`. It has one data members `radius`.

The data member is declared private and access is provided by declaring set and get methods.

Solution:

```
public class Circle
{
    Private int radius;
    public Circle()
    {
        radius=7;
    }
    public Circle(int r)
    {
        radius=r;
    }

    public void setRadius(int r)
```

```

    {
    radius=r;
    }
    Public int getRadius(){
        return radius;
    }

    public void display()
    {
    System.out.println("radius = "+radius);
    }
    public void CalculateCircumfrance(){
    double a=3.14*radius*radius;
    System.out.println("Circumfrance = "+a);
    }
    }

    Public class Runner
    {
    Public static void main ()
    {
    Circle c = new Circle();
    c1.setRadius(5);
    System.out.println("circumference of Circle 1 is " + c1.area( ));

    Int r = c1.getRadius();

    Circle c2 = new Circle(r);
    c2.setRadius(5);
    System.out.println("circumference of Circle 2 is " + c2.area( ));

    }

    }

```

Activity 2

The following example shows the declaration of class Rectangle. It has two data members that represent the length and width of rectangle. Both data member are declared private and access is provided by declaring set and get methods for both data members.

Solution:

```

public class Rectangle

```



```

{

    Private int length, width;

    public Rectangle()
    {
        length = 5;   width = 2;
    }

    public Rectangle(int l, int w)
    {
        length = l;   width = w;
    }
    public void setLength(int l) //sets the value of length
    {
        length = l;
    }

    public void setWidth(int w) //sets the value of width
    {
        width = w;
    }

    public void getLength() //gets the value of length
    {
        return length;
    }

    public void getWidth() //gets the value of width
    {
        return width;
    }

    Public int area ()
    {
        return (length*width);
    }
}

```

```

Public class Runner
{
    Public static void main ()
    {
        Rectangle rect = new Reactangle();
        Rect.setLength(5);
        Rect.setWidth(10);
    }
}

```

```

        System.out.println("Area of Rectangle is " +
        rect.area()); System.out.println("width of Rectangle
        is " + rect.getWidth());
    }
}

```

3. Stage ▼ (verify)

Home Activities

Activity 1

Create an Encapsulated class Marks with three data members to store three marks. Create set and get methods for all data members. Test the class in runner

Activity 2:

Create an Encapsulated class Account class with balance as data member. Create two constructors and methods to withdraw and deposit balance. In the runner create two accounts. The second account should be created with the same balance as first account. (hint: use get function)

Activity 3.

Write a class Circle, which will model the functionality of a Circle.

1. Attributes
 - ▶ radius
2. Methods
 - ▶ Setter function for radius
 - ▶ Getter function for radius
 - ▶ To compute area
 - ▶ To compute perimeter

Note: For value of pi, use Math.PI

Expected Output:

```

E:\>java CircleTest
Enter radius of circle:5
Radius of circle is 5.00
Area is 78.50
Perimeter is 31.40

```

Activity 4

Make a Rectangle class that has color, width and height attribute. Color is of String type, while other two are int type attribute. All the attributes should be private and exposed via setter/getter methods. Define a method inside Rectangle class: *int calculateArea()* that returns area of Rectangle. Define another method in Rectangle class: *int calculatePerimeter()* that returns perimeter of rectangle. Make a RectangleTest class, in main method, instantiate a rectangle object, sets its width and height, and prints its area and perimeter.

Activity 4

Create **GardeBook** class that contains **instructorName** and **courseTitle** private attributes. Define getter and setter methods for both attributes, and another method named void *displayGradeBookInfo()*. When this method is called, it should print courseTitle and instructorName.

Make another class named GardeBookTest, in its main method, instantiate GradeBook object and call *displayGradeBookInfo()* method.

Note: Class variables should be write-only.

Expected Output:

```
run:
Course Name is:Lab Object Orient Programming
Course Name is: Muntaha Iqbal
```

4. Stage a2 (access)

Assignment

Task 1:

Suppose you operate several hot dog stands distributed throughout town. Define an Encapsulated class named **HotDogStand** that has an instance variable for the hot dog stand's. ID number and an instance variable for how many hot dogs the stand has sold that day.

Create a constructor that allows a user of the class to initialize both values. Also create a method named justSold that increments by one the number of hot dogs the stand has sold. The idea is that this method will be invoked each time the stand sells a hot dog so that you can track the total number of hot dogs sold by the stand.

Write a main method to test your class with at least three hot dog stands that each sell a variety of hot dogs. Use get function to display the hot dogs sold for each object.

Task 2:

Write a class BankAccount, which have following attributes:

1. Attributes

- ▶ accountNumber
- ▶ accountTitle

Both attributes are private.

Create a test class named BankAccountTest. In main method, instantiate BankAccount object and display information of account.

Note: Class variables should be read-only.

Expected Output:

```
run:
Account Number: 23057000086123
Account Title: Muhammad Ali
BUILD SUCCESSFUL (total time: 0 seconds)
```

Statement Purpose:

This Lab will help students in understanding the concept static data and member function of a class.

Activity Outcomes:

After completion of this Lab students know in which scenario static data members, static functions and static objects can are used.

Instructor Note:

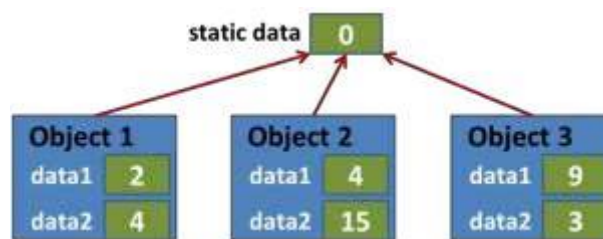
The student should have understanding about access specifiers.

The students should understand **the concept of object's address space in memory.**

1. Stage J (Journey)

Introduction

There is an important exception to the rule that each object of a class has its own copy of all the data members of the class. In certain cases, only one copy of a variable should be shared by all objects of a class. A **static data member** is used for such propose.. Such a variable represents "class-wide" information (i.e., a property of the class shared by all instances, not a property of a specific object of the class). The declaration of a static member begins with keyword static.



In the above figure, consider Object1, Object2 and Object3 are three of a class. Each object has its own private data members. The class has a static data member that is share among all the object of that class. This data member has it own memory allocated that is separate from the memory allocated for each object of that class.

Static data member – access

A public static data member can be access using any object of the class or class name

- [Object Name].[name of static variable]
 - e1.count;
- [Class Name] .[name of static variable]
 - Employee .count;

A private static member can be access using a public member function

- [objectName].[public member function]
 - e1.getCount();
- [Class name].[public member function]
 - Employee.getCount();

A static method is a method that can only access static data member of class. It cannot access non static data member of that class

Static vs. non static functions

- Static function can only access static data member of class
 - className.static_ method ();
 - Object_of_class.static_method();

- Non static member function can access static and non static data member of class
 - Object_of_class.Non_static_method ();

2. Stage a1 (apply)

Lab Activities

Activity 1:

The following example demonstrates that static variables are common for all instances:

Solution:

```
Public class ABC
{
    static int Var1=77;           //Static integer variable
    String Var2;                 //non-static string variable
}
Public class ABCRunner
{
    public static void main(String args[])
    {
        ABC ob1 = new ABC ();
        ABC ob2 = new ABC ();
        ob1.Var1=88;
        ob1.Var2="I'm Object1";
        ob2.Var2="I'm Object2";
        System.out.println("ob1 integer:"+ob1.Var1);
        System.out.println("ob1 String:"+ob1.Var2);
        System.out.println("ob2 integer:"+ob2.Var1);
        System.out.println("ob2 STring:"+ob2.Var2);
    }
}
```

Output:

```
ob1 integer:88
ob1 String:I'm Object1
ob2 integer:88
ob2 String:I'm Object2
```

Activity 2:

The following example demonstrates counting of objects by using static variable.

Solution:

```
Public class NoOfObjects {

    Private static int objs=0;
    Private int a;

    Public NoOfObjects(){
        objs++;
    }

    Public NoOfObjects(intx){
        a=x;
        objs++;
    }

    Public static int getObjs (){
        return objs;
    }
}

Public class NoOfObjectsRunner
{

    Public static void main(String[] args){

        NoOfObjects o1=newNoOfObjects();
        NoOfObjects o2=newNoOfObjects(122);
        NoOfObjects o3=newNoOfObjects(150);

        System.out.println("Objects created:"+
        NoOfObjects.getObjsCreated()); System.out.println("Objects
        created:"+ o1.getObjsCreated()); }

    }
```

Activity 3

The following example demonstrates static methods declaration.


```

Public class ABC
{
    Public static int i;
    public String s;

    Public static void displayStatic() //Static method
    {
        System.out.println("i:"+i);
    }
    Public void display()           //non static method
    {
        System.out.println("i:"+i);
        System.out.println("s:"+s);
    }
}
Public class ABCRunner
{
    public static void main(String args[]) //Its a Static Method
    {
        ABC a = new ABC();
        a.display();
        ABC.displayStatic();
    } }

```

3. Stage v (verify)

Home Activities

Activity 1

Create a SavingsAccount class. Use a static data member annualInterestRate to store the annual interest rate for each of the savers. Each member of the class contains a private data member savingsBalance indicating the amount the saver currently has on deposit. Provide member function calculateMonthlyInterest that calculates the monthly interest by multiplying the balance by annualInterestRate divided by 12; this interest should be added to savingsBalance.

Provide a static member function modifyInterestRate that sets the static annualInterestRate to a new value.

Write a driver program to test class SavingsAccount. Instantiate two different objects of class SavingsAccount, saver1 and saver2, with balances of \$2000.00 and \$3000.00, respectively. Set the annualInterestRate to 3 percent. Then calculate the monthly interest and print the new balances for each of the savers.

Then set the annualInterestRate to 4 percent, calculate the next month's interest and print the new balances for each of the savers.

Activity 2

Write program to count the number of objects created and destroyed for a class using static data members and static member functions

Write the output of following programs:

Activity 3

Write the output of following programs:

Program 1

StaticVariable.java

```
public class StaticVariable {  
    int i=0;  
  
    public StaticVariable() {  
        i++;  
        System.out.printf("Value of i is %d\n",i);  
    }  
}
```

StaticVariableTest.java

```
public class StaticVariableTest {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        // TODO code application logic here  
        StaticVariable obj1=new StaticVariable();  
        StaticVariable obj2=new StaticVariable();  
        StaticVariable obj3=new StaticVariable();  
        StaticVariable obj4=new StaticVariable();  
    }  
}
```

Program 2

StaticVariable.java

```
public class StaticVariable {  
    static int i=0;  
  
    public StaticVariable() {  
        i++;  
        System.out.printf("Value of i is %d\n",i);  
    }  
}
```

StaticVariableTest.java

```
public class StaticVariableTest {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        // TODO code application logic here  
        StaticVariable obj1=new StaticVariable();  
        StaticVariable obj2=new StaticVariable();  
        StaticVariable obj3=new StaticVariable();  
        StaticVariable obj4=new StaticVariable();  
    }  
}
```

Activity 4

Fix the errors of following program:

Program 1:

StaticMethods.java

```
public class StaticMethods {  
    public static void show()  
    {  
        System.out.println("Static Method in class StaticMethod");  
    }  
}
```

StaticMethodsTest.java

```
public class StaticMethodsTest {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void display()  
    {  
        System.out.println("Static Method in class StaticMethodTest");  
    }  
    public static void main(String[] args) {  
        // TODO code application logic here  
        display();  
        show();  
    }  
}
```

```
}  
  
}
```

Program 2:

StaticMethods.java

```
public class StaticMethods {  
    public int i=10;  
    public static void show()  
    {  
        System.out.printf("Value of i is %d\n",i);  
    }  
}
```

StaticMethodsTest.java

```
public class StaticMethodsTest {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void display()  
    {  
        System.out.println("Static Method in class StaticMethodTest");  
    }  
    public static void main(String[] args) {  
        // TODO code application logic here  
        display();  
        show();  
    }  
}
```

Program 3:

StaticMethods.java

```
public class StaticMethods {  
  
    public static void show()  
    {  
        System.out.printf("Static Method in class StaticMethods");  
        display();  
    }  
}
```

```

    public void display()
    {
        System.out.printf("Non-static Method in class StaticMethods");
    }
}

```

StaticMethodsTest.java

```

public class StaticMethodsTest {

    /**
     * @param args the command line arguments
     */
    public static void display()
    {
        System.out.println("Static Method in class StaticMethodTest");
    }
    public static void main(String[] args) {
        // TODO code application logic here
        display();
        show();
    }
}

```

4. Stage a2 (access)

Assignment

Task 1:

Make a class Student, which has following attributes:

1. ID
2. Name
3. CGPA
4. UniversityName (static variable)

Now create a method *displayInformation()*, which prints id, name, cgpa and university name of students. You also have to create parameterized constructor.

Create a test class StudentTest and instantiate four student objects with following name, Ali, Ahmad, Tooba and Fatima and for each object call *displayInformation()* method.

Expected Output:

```
run:
ID:SP18-BSE-001 Name:ALI CGPA: 3.54 Univeristy:COMSATS
ID:SP18-BSE-002 Name:Ahmad CGPA: 2.54 Univeristy:COMSATS
ID:SP18-BSE-003 Name:Tooba CGPA: 3.0 Univeristy:COMSATS
ID:SP18-BSE-004 Name:Fatima CGPA: 3.12 Univeristy:COMSATS
```

Task 2:

Make a class Employee. Employee class has one static variable count (initialize it to zero), and two attributes firstName and lastName. firstName and lastName are private. (Variable count maintains a count of the number of object of class Employee). Create a parameterized constructor. In this constructor, you have to increment the count and print the information of employee. Create a static method *int employyeCount()* which returns count of employees.

Create a test class with EmployeeTest. In main method, instantiate two objects (obj1, obj2) of employees. You have to print number of employees before instantiation and after instantiation. Also, print the first name of both employees.

Note: No need to create extra getters and setters.

Task 3:

- Write a JAVA program that creates a class MyGame, where class attributes are word1, word2 and word3 (String type) and static variable score(int)
- In MyGame create a static method, level1(with 3 string type arguments) which compare the given 3 words in arguments from a word given by user.
- If user input = word 1, score will have -1; if user input = word 2, score will have +5 and if user input = word3, score will have +1
- Now make another class LevelTwo which will have another static method leveltwo which will give bonus of 10 and print the score and also print "You are Now at LEVEL 2"
- In main (GameTest Class) leveltwo will be loaded only when score value will be 5

Statement Purpose:

Objective of this lab is to make students to input form user in java.

Activity Outcomes:

The student will be able to use Scanner class in Java

Instructor Note:

The Students should have of Classes and objects.

1. Stage J (Journey)

Introduction

Interactive program: Reads input from the console. While the program runs, it asks the user to type input. The input typed by the user is stored in variables in the code. Can be tricky; users are unpredictable and misbehave. However, interactive programs have behavior that is more interesting.

The Scanner class is found in the java.util package.

```
import java.util.*; // so you can use Scanner
```

Constructing a Scanner object to read console input:

```
Scanner console = new Scanner(System.in);
```

```
class Scanner{
    InputStream source;

    Scanner(InputStream src){
        this.source = src;
    }

    int nextInt(){
        int nextInteger;
        //Scans the next token of the input as an int from the source.
        return nextInteger;
    }
}
```

With the parameter System.in. That means it is going to read from the standard input stream of the program. Communicates with System.in (the opposite of System.out)

2. Stage **a1** (apply)

Lab Activities

Activity 1

The following example demonstrates static methods declaration.

Solution

```
package scannerexample;
import java.util.Scanner;
public class Student {
```



```

private int age;
private String name;
private double cgpa;

public Student() {
}
public Student(int age,String name,double cgpa) {
    this.age = age;
    this.name=name;
    this.cgpa=cgpa;
}
/*
public Student(int age, String name, double cgpa) {
    //this.age = age;
    //this.name = name;
    //this.cgpa = cgpa;
    setAge(age);
    setName(name);
    setCgpa(cgpa);
}
*/

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public double getCgpa() {
    return cgpa;
}

public void setCgpa(double cgpa) {
    this.cgpa = cgpa;
}

public void input()
{
    Scanner sc=new Scanner(System.in);
    Scanner sc1=new Scanner(System.in);
    System.out.print("Enter Age:");

```

```

        this.age=sc.nextInt();
        System.out.print("Enter name:");
        this.name=sc1.nextLine();
        System.out.print("Enter cgpa:");
        this.cgpa=sc.nextDouble();

    }

}
package scannerexample;
import java.util.Scanner;

/**
 *
 * @author AC
 */
public class ScannerExample {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner sc=new Scanner(System.in);
        Scanner sc1=new Scanner(System.in);

        System.out.println("Enter Age:");
        int age=sc.nextInt();
        System.out.println("Enter Name:");
        sc.next();
        String name=sc.nextLine();
        //String name=sc1.nextLine();
        System.out.println("Enter CGPA:");
        double cgpa=sc.nextDouble();
        Student obj1=new Student(age,name, cgpa);

        Student s1=new Student();
        s1.input();
        System.out.println("Name:"+s1.getName()+" Age:"+s1.getAge()+" CGPA:"+s1.getCgpa());
    }
}

```

3. Stage ▼ (verify)

Home Activities

Activity 1

You have to write a “Very simple calculator” program. The program asks the user for two numbers, and then asks for the operation to be performed on those numbers (assume only arithmetic operations).

Options: Enter 1 for addition, 2 for multiplication, 3 for subtraction, 4 for division, 5 for mod

Hint: 1. Usage of scanner object for taking input, 2. Usage of switch statement

Expected Output:

```
Enter First Number:24
Enter Second Number:5
Enter 1 for addition
2 for multiplication
3 for subtraction
4 for division
5 for mod
Enter your choice:5
4
```

Activity 2

Write a program in java, which inputs user name, ID, semester, section, GPA, CGPA and marks in Programming Fundamentals and print the information in following format:

Expected Output:

Activity 3

Write a program in java, which inputs three numbers from user and prints largest on screen.

```
Enter First Number:100
Enter Second Number:10
Enter third Number:2
100 is largest number
```

4. Stage a2 (access)

Assignment

Task 1:

BankAccount class should be in package **com.oopLab.userInput** and BankAccountTest class should be in package **com.oopLab.userInput.test**

Make BankAccount class with balance and name attributes of type double and String. Define public void deposit(double amount) and public void withdraw(double amount) methods. Deposit should increase the balance by passed value and withdraw should decrease the balance with passed amount.

Make BankAccountTest class. In main method, create a new object of BankAccount class.

Get balance and name of account holder from user input and initialize both object attributes.

Then show this menu:

Press 1: To Deposit an amount

Press 2: To Withdraw an amount

Press 3: To View the current balance

If user press 1: show following:

Enter the amount you want to deposit in your account >

For example. If user enters 500, call the deposit method of BankAccount object and pass 500 to it. Do it for option 2 but call the withdraw method. If user choose 3 from menu, print the current balance.

Statement Purpose:

Objective of this lab is to make students understand the difference between object oriented and procedural approaches to programming

Activity Outcomes:

The student will understand the advantages of using OOP

The student will understand the difference between procedural and object oriented approaches

Instructor Note:

The Students should have knowledge about structured programming.

1. Stage J (Journey)

Introduction

Enum is used to define fixed set of constants. Without enum, we define constants as:

```
static final int WON = 1;
static final int LOST = 2;
static final int CONTINUE = 3;
```

Issues:

- Unsafe - invalid values may be passed or returned
- Inconvenient to iterate, print lists, get subset, etc.

Enum is a new class type introduced in Java 1.5. Enum example: enum GameStatus {LOST, WON, CONTINUE}. It contains fixed set of constants e.g. Game Status, Days, Planets, Gender, etc. Provides compile-time type safety. These constants implicitly **public, static and final**. Enum has many things common with classes, but we **cannot create new object of enum** because enum **constructor is private** by default. Each constant is actually a **reference to an object of that enum type**.

2. Stage a1 (apply)

Lab Activities

Activity 1

The following example demonstrate the use of Enum in Java.

Solution

```
public enum GameStatus{
    WON, LOST, CONTINUE;
}

// Below code is approximation
public enum GameStatus extends java.lang.Enum {

    public static final GameStatus WON = new GameStatus("WON", 0);
    public static final GameStatus LOST = new GameStatus("LOST", 1);
    public static final GameStatus CONTINUE = new GameStatus("CONTINUE", 2);

    private GameStatus(String name, int ordinal){
        this.name = name;
        this.ordinal = ordinal;
        // can be accessed using name() and ordinal() methods
    }

    public static GameStatus[] values(){
        return new GameStatus[]{WON, LOST, CONTINUE};
    }
}
```

Activity 2

Here is another example of Enum in Java.

Solution

```
public enum Planet {
    MERCURY(3.302e+23, 2.439e6),
    VENUS (4.869e+24, 6.052e6),
    EARTH (5.975e+24, 6.378e6),
    MARS (6.419e+23, 3.393e6),
    JUPITER(1.899e+27, 7.149e7),
    SATURN (5.685e+26, 6.027e7),
    URANUS (8.683e+25, 2.556e7),
    NEPTUNE(1.024e+26, 2.477e7);

    private final double mass; // In kilograms
    private final double radius; // In meters
    private final double surfaceGravity; // In m / s^2

    // Universal gravitational constant in m^3 / kg s^2
    private static final double G = 6.67300E-11;

    // Constructor
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
        surfaceGravity = G * mass / (radius * radius);
    }

    public double mass() { return mass; }
    public double radius() { return radius; }
    public double surfaceGravity() { return surfaceGravity; }
    public double surfaceWeight(double mass) {
        return mass * surfaceGravity; // F = ma
    }
}

public class PlanetTest{

    public static void main(String args[]) {
        double earthWeight = Double.parseDouble(args[0]);
        double mass = earthWeight / Planet.EARTH.surfaceGravity();
        for (Planet p : Planet.values())
            System.out.printf("Weight on %s is %f\n", p, p.surfaceWeight(mass));
    }
}
```

3. Stage v (verify)

Home Activities

Activity 1

Make an enum of Gender, which has two possible values, “Female” and “Male”.

Make a class **Employee** that has four attributes, id, name, gender and basicSalary. Use above enum for gender. Make a fully parameterized constructor.

- a. This class has a method *calculateBonus()* that calculate bonus using following rules:
 - i. If employee is male, he will get 10% bonus on his basic salary.
 - ii. If the employee is female, she will get 12% bonus on her basic salary.

Make a class EmployeeTest and create an array of Employee to input information of all employees. Ask user to input number of employees. All attributes values of Employee (id,name,basicSalary and gender) shall be taken from user.

Expected Output:

```
Enter number of employees:2
Enter Employee Id:1
Enter Employee Name:Ali
Enter Employee Basic Salary:60000
Enter Employee Gender:Male
Bonus for Employee is: 6000.0
Enter Employee Id:2
Enter Employee Name:Tooba
Enter Employee Basic Salary:80000
Enter Employee Gender:Female
Bonus for Employee is: 9600.0
BUILD SUCCESSFUL (total time: 24 seconds)
```

4. Stage a2 (access)

Assignment

Task 1:

Make an enum for Book. Book is an enumeration of four values i.e. JHTP, CHTP, CPPHTP, VBHTP. Each constant is followed by two values title and copyrightYear. Title and copyrightYear for each constant are given below:

JHTP JAVA HOW TO PROGRAM 2010

CHTP C HOW TO PROGRAM 2011

CPPHTP C++ HOW TO PROGRAM 2012

VBHTP VISUAL BASIC HOW TO PROGRAM 2013

This enumeration has a constructor, which has two values title and copyrightYear, two instance variables (both variables are private) and two getter methods.

Create a class EnumTest. In this class first display information of all books. In addition, display a range of enum constants (from CHTP to VBHTP).

Books Information:

JHTP JAVA How to Program 2010
CHTP C how to Program 2011
CPPHTP C++ how to Program 2011
VBHTP Visual Basic how to Program 2013

Some Books Information :

CHTP C how to Program 2011
CPPHTP C++ how to Program 2011
VBHTP Visual Basic how to Program 2013

Statement Purpose:

The purpose of lab is to make students understand the concept of has-a relationship in object oriented programming.

Activity Outcomes:

Students will be able to understand that complex objects can be modeled as composition of other objects.

Students will be able to implement programs related to composition.

Instructor Note:

The student should be able to declare a class with primitive data members.

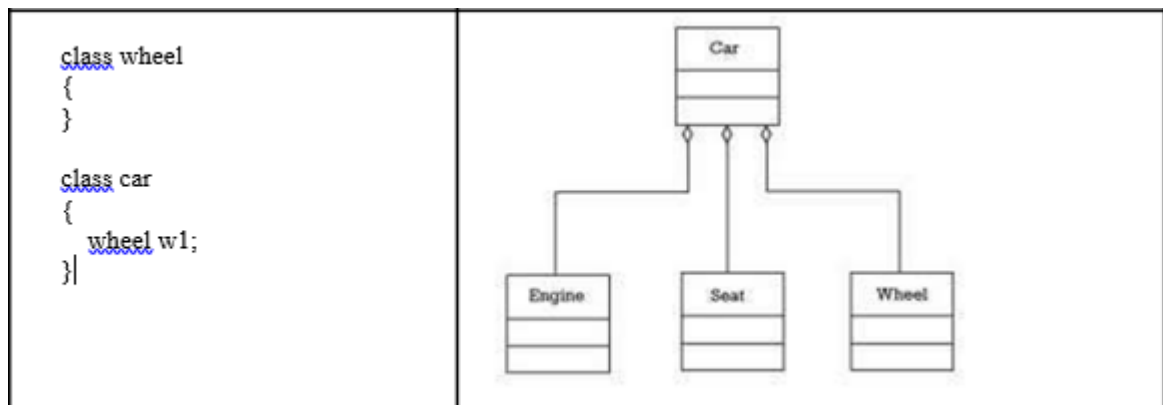
The student should be able to define methods; that can receive and return objects.

1. Stage J (Journey)

Introduction

In real-life, complex objects are often built from smaller, simpler objects. For example, a car is built using a metal frame, an engine, some tires, a transmission, a steering wheel, and a large number of other parts. A personal computer is built from a CPU, a motherboard, some memory, etc. Even you are built from smaller parts: you have a head, a body, some legs, arms, and so on. This process of building complex objects from simpler ones is called composition (also known as object containership).

More specifically, composition is used for objects **that have a “has-a” relationship to each** other. A car has-a metal frame, has-an engine, and has-a transmission. A personal computer has-a CPU, a motherboard, and other components. You have-a head, a body.



2. Stage a1 (apply)

Lab Activities

Composition is about expressing relationships between objects. Think about the example of a manager. A manager has the properties e.g Title and club dues. A manager has an employment record. And a manager has an educational record. The phrase "has a" implies a relationship where the manager owns, or at minimum, uses, another object. It is this "has a" relationship which is the basis for composition. This example can be programmed as follows:

Solution:

```

public class studentRecord
{
    Private String degree;

    public studentRecord()
    {}
    public void setDegree ( String deg)
    {
        degree = deg;
    }

    public void getDegree ()
    {
        return degree;
    }
}

class employeeRecord
{
    Private int emp_id;
    Private double salary;
    public employeeRecord ()
    {}
    public void setEmp_id ( int id)
    {
        emp_id = id;
    }
    public intgetEmp_id ()
    {
        return emp_id ;
    }
    public void setSalary ( intsal)
    {
        Salary = sal;
    }
    public StringgetSalary ( intsal)
    {
        Salary = sal;
    }
}

class Manager
{
    Private String title;

```

```

    Private double dues;
    Private employeeRecord emp;
    Private studentRecord stu;

    Public manager(String t,double d,employeeRecord e,studentRecord s)
    {
        title = t;
        dues = d;
        emp = e;
        stu = s; }

    Public void display()
    {
        System.out.println("Title is : " + title);
        System.out.println("Dues are : " + dues);

        System.out.println("Employee record is as under:");
        System.out.println("EmployeeId is : " + emp.getEmp_id());
        System.out.println("EmployeeSalary is : " + emp.getSalary());

        System.out.println("Student record is as
        under:"); System.out.println("Degree is :
        " + stu.getDegree()); }
}
Public class Runner
{
    void main()
    {
        studentRecord s = new studentRecord("MBA");
        employeeRecord e = new employeeRecord(111, 50000);
        Manager m1 = new Manager("financeManager ", 5000, e, s );
        m1.display();
    } }

```

Activity 2

Think about the example of an employee. A manager has the properties e.g name and address. An employee has address. The phrase "has a" implies a relationship where the employee owns, or at minimum, uses, another object. It is this "has a" relationship which is the basis for composition. This example can be programmed as follows:

Solution:

```

public class Address {
    private String city;
    private String country;
    public Address(String city, String country) {
        this.city = city;
        this.country = country;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }
}

public class Person {
    private String name;
    private Address add;

    public Person(String name, Address add) {
        this.name = name;
        this.add = add;
        add=new Address("Lahore","Pakistan");
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {

```

```

        this.name = name;
    }
    public Address getAdd() {
        return add;
    }

    public void setAdd(Address add) {
        this.add = add;
    }

    public String getPersonCity()
    {
        return add.getCity();
    }
}
public class CompositionClassExample {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Address obj=new Address("Lahore","Pakistan");
        Person p=new Person("Ali",obj);
        Person p2=new Person("Ahmad",new Address("Isb","Pakistan"));
        System.out.print(p.getName());
        System.out.println(" "+p.getAdd().getCity());
        //p.setAdd(new Address("Bhp","Pakistan"));
        //obj.setCity("Bhp");
        p.getAdd().setCity("Bhp");
        System.out.print(p.getName());
        p.setAdd(obj);
        System.out.println(" "+p.getAdd().getCity());

        System.out.println(p.getPersonCity());

    }
}

```

```
}
```

3. Stage v (verify)

Home Activities

Activity 1

Make a class **Course** with two attributes, `courseCode` and `courseTitle`. Access modifier for both attributes is `public`. This class has fully parameterized constructor

Make a class **Student** that has 2 public attributes, `ID`, and `Name` and another attribute `course` of type `Course`. `Course` attribute is `private`. This class has two constructors, no argument constructor and one with `studentID`.

Make another class **StudentTest** and print `courseTitle` and `Id` of student.

Activity 2

Make a class **Address** that have three attributes, `streetNumber`, `city` and `country`. All attributes are `private` and will be exposed using getter and setter method.

Make an **Employee** class, which have `employeeID`, `employeeName` and `employeeAddress`. All attributes are `private`. Attributes `employeeAddress` should be of type `Address`. In this class, define a constructor, which takes two parameters i.e. `employeeID` and `employeeAddress`.

Make another class **EmployeeTest**, in main method, instantiate an employee object named `employee1`. Initialize name and address attribute using constructor. The attribute name should be initialized using setter method. User should give value of name attribute. Also, print values of all attributes using getter.

Make another employee object named `employee2`. Both employees share same address. Print the id and address for `employee2`.

Expected Output:


```

run:
Employee 1 ID:123
Enter name of first employee
Ali
Employee 1 Name:Ali
Employee 1 Address:
Johar Town, S#10
Lahore
Pakistan
Employee 2 ID:456
Employee 2 Address:
Johar Town, S#10
Lahore
Pakistan
BUILD SUCCESSFUL (total time: 3 seconds)

```

Activity 3

Make a class **Date** with three attributes, day, month and year. All attribute are private. In constructor, you have to validate month. If month is out of range, you have to print a message, “Invalid Month”. Suppose all months have 30 days.

Make a class **Employee** that has four instance variable firstName, lastName, birthdate and hiringDate. firstNmae and lastName are reference to String object while birthdate and hiringDate are references to Date object. All instance variable are private. Employee class has a fully parameterized constructor.

Make another class **EmployeeTest**. In main method, instantiate 2 Date object to represent birthdate and hiringDate of employee. Now instantiate one object for Employee named employee1 and initialize firstName and lastName using constructor. Print values of all attributes using getter method.

Expected Output:

```

run:
First Name of Employee:Muhammad
First Name of Employee:Ali
Birth Date:20:12:1995
Hiring Date:22:9:2019
BUILD SUCCESSFUL (total time: 0 seconds)

```

4. Stage a2 (access)

Assignment

Task 1:

For each class/attributes described below, choose appropriate data type. All attributes of each class should be private and exposed via get/set methods. Also define at least one constructor shall take and initialize 2-3 attributes of the object.

1. Define a class **Course** with courseCode and courseTitle attributes.
2. Define a class **PhoneNumber** with countryCode, cityCode and lineNumber attributes.
3. Define a class **Address** with streetAddress, town, city, country and phoneNumber attributes.
Attribute phoneNumber shall be of type PhoneNumber.
4. Define a class **Student** with name, email, cnic, course1, course2 and address attributes.
Where course1 and course2 should be of type Course and address shall be of type Address.
Define a constructor in Student class that takes cnic, name and address only.

Create a **StudentTest** class, in its main method, create a Student object i.e. student1. Fully initialize its' all attributes. cnic, name and address shall be initialized by constructor, other attributes shall be initialized using setter methods. All attributes values shall be taken from user. After the object is fully initialized, print all, attribute values from student object reference.

Make another object student2, assume the student live at same address as student1. Reuse the address object of student1 to initialize student2 address. You do not need to take attributes from user input for student2 object. Change some attribute of address from student1 and check, does it also change for student2, understand why and why not?

Expected Output:

```

CompositionTest (run) × CompositionUpdated (run) ×
run:
Name:Ali
CNIC:31200-5212987-3
Street#10 Johar Town Lahore Pakistan
Enter Email: muhammadali123@gmail.com
Email:muhammadali123@gmail.com
Enter Course title : OOP
Enter code: CSC203
Enter Course title : PF
Enter code: CSC103
Course Code:OOP
Course Title:CSC203
Course Code:PF
Course Title:CSC103
Student 2 Data:
Name:Tooba
CNIC:31204-7636901-2
Street#10 Johar Town Lahore Pakistan
BUILD SUCCESSFUL (total time: 27 seconds)
|

```

Statement Purpose:

Objective of this lab is to make students understand the usage of primitive and reference array in java

Activity Outcomes:

The student will understand the difference primitive and reference arrays.

Instructor Note:

The Students should have knowledge about arrays data structure.

1. Stage J (Journey)

Introduction

Java arrays are ordered collections of primitive, object reference (In java arrays are objects). Java arrays are homogeneous – all elements of an array must be of same type. Three steps to create and use an array:

1. Declaration
2. Construction
3. Initialization

Declaration tell compiler the array name and its type of its elements

```
int nums[];  
double dobs[];  
Student stds[];  
float[][] twoDim;
```

So declaration does not requires size of the array. Size is specified at runtime, when array is allocated via new keyword ... we call it array construction. Declaration and construction can be performed in a single line too:

```
int[] nums = new int[size];
```

2. Stage a1 (apply)

Lab Activities

Activity 1

The program below represents an example of array in java.

Solution

```
package arrayclassexample;  
  
import java.util.Scanner;  
  
public class ArrayClassExample {  
  
    public static void main(String[] args) {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enster size of array");
```

```

int size=sc.nextInt();

int sum=0;

int array[]=new int[size];

//for(int i=0;i<array.length;i++)

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

{

    array[i]=sc.nextInt();

    sum=sum+array[i];

}

System.out.println("Sum is:"+sum);

}

}

```

Activity 2

Example of reference type Array is given below

Solution

```

public class Student {
    private int id;
    private String name;
    private double cgpa;

    public Student() {
    }

    public Student(int id, String name, double cgpa) {
        this.id = id;
        this.name = name;
        this.cgpa = cgpa;
    }
}

```

```

public void display()
{
    System.out.println("Id: "+id+" Name: "+name+" CGPA:"+cgpa);
}

}

public class ArrayClassExample1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner sc=new Scanner(System.in);
        Scanner sc1=new Scanner(System.in);
        System.out.print("Enter total no of student:");
        int size=sc.nextInt();
        Student s[]=new Student[size];
        //Student s1;
        // s1=new Student();
        for(int i=0;i<size;i++)
        {
            System.out.print("Enter id:");
            int id=sc.nextInt();
            System.out.print("Enter name:");
            String name=sc1.nextLine();
            System.out.print("Enter cgpa:");
            double cgpa=sc.nextDouble();
            s[i]=new Student(id,name,cgpa);
            // s1=new Student(id,name,cgpa);

        }
        //s1.display();
    }
    /**
     for(int j=0;j<s.length;j++)
     {
         s[j].display();
     }

```

```

    }
*/
    for(Student a:s)
    {
        a.display();
    }

}

}

```

3. Stage v (verify)

Home Activities

Activity 1

Make an array of n elements and initialize it, data and size of the array shall be given by user. Print all elements, print minimum number and its index in array, print maximum number and its index in the array and also print the average number.

Activity 2

Roll a 6 faces dice 1 million times, print how many times each face appear along with its percentage? Use arrays to store each face count e.g. face[3] shall store how many times 4 appeared during random rolls. See figure 7.7 of book for detail. (From Java How to Program 10th)

Activity 3

Make a program that shall shuffle cards. Print the deck before shuffling and after shuffling. See Figure 7.9, 7.10 and 7.11 from the book for detail. (From Java How to Program 10th)

4. Stage a2 (access)

Assignment

Task 1:

Extend Activity 2 (from Lab 7), such that a Student may have more than one address i.e. you shall define addresses attribute in Student class that shall be an array of Address. In main method, create an array of Student that represent all students in a classroom. Let user input the number of students and student's attributes (id and name shall be enough). Let user enter the number of addresses of each student need to store. Get address attributes from user input too. Print all the information on console after the objects are initialized.

Statement Purpose:

Objective of this lab is to make students understand the usage of multidimensional arrays in java.

Activity Outcomes:

The student will be able understand the multidimensional arrays.

Instructor Note:

The Students should have knowledge one-dimensional primitive and reference arrays in java.

1. Stage J (Journey)

Introduction

One-D array only held one column of data. However, you can set up an array to hold more than one column. These are called multi-dimensional arrays.

Syntax is very simple:

```
int nums[][] = new int[3][4];
```

We have declared and constructed the array

MD arrays are array of arrays. So 3 indicates the 'nums' array is of size 3 (3 rows). And 4 indicates, each element points to any array of size 4 (4 columns)

nums length is 3, not 12 (you must understand, why?)

nums[0] length is 4

2. Stage a1 (apply)

Lab Activities

Activity 1

The program below represents an example of multidimensional array in java.

Solution

```
public static void main(String[] args) {  
  
    int[][] aryNumbers = new int[6][5];  
  
    aryNumbers[0][0] = 10;    aryNumbers[1][0] = 20;  
    aryNumbers[0][1] = 12;    aryNumbers[1][1] = 45;  
    aryNumbers[0][2] = 43;    aryNumbers[1][2] = 56;  
    aryNumbers[0][3] = 11;    aryNumbers[1][3] = 1;  
    aryNumbers[0][4] = 22;    aryNumbers[1][4] = 33;  
  
    aryNumbers[2][0] = 30;    aryNumbers[3][0] = 40;  
    aryNumbers[2][1] = 67;    aryNumbers[3][1] = 12;  
    aryNumbers[2][2] = 32;    aryNumbers[3][2] = 87;  
    aryNumbers[2][3] = 14;    aryNumbers[3][3] = 14;  
    aryNumbers[2][4] = 44;    aryNumbers[3][4] = 55;
```

```

aryNumbers[4][0] = 50;      aryNumbers[5][0] = 60;
aryNumbers[4][1] = 86;      aryNumbers[5][1] = 53;
aryNumbers[4][2] = 66;      aryNumbers[5][2] = 44;
aryNumbers[4][3] = 13;      aryNumbers[5][3] = 12;
aryNumbers[4][4] = 66;      aryNumbers[5][4] = 11;

int rows = 6;
int columns = 5;
int i, j;

for ( i = 0; i < rows; i++) {
    for ( j = 0; j < columns; j++) {
        System.out.print(aryNumbers[i][j] + " ");
    }
    System.out.println( "" );
}

```

Activity 2

Another program of multidimensional array in java.

Solution

```

package mdarrayclassexample2;

import java.util.Scanner;

/**
 *
 * @author AC
 */
public class MDArrayClassExample2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter size of array:");
        int size=sc.nextInt();
        int arr[][]=new int[size][];
    }
}

```

```

for(int i=0;i<arr.length;i++)
{
    System.out.print("Enter size of array "+(i+1)+":");
    arr[i]=new int[sc.nextInt()];
    System.out.print("Enter elemets of array:");
    for(int j=0;j<arr[i].length;j++)
    {
        arr[i][j]=sc.nextInt();
    }
}
for(int a[]:arr)
{
    for(int b:a)
    {
        System.out.print(b+" ");
    }
    System.out.println();
}
}
}

```

3. Stage v (verify)

Home Activities

Activity 1

Make StudentTest class which has 2-dimensional int array of marks.

i.e.

```
int[][] marks=new int[noOfStudents][];
```

First user will input total number of students and then number of test for each student.

Below array is just an example, it contains marks of two students, 1st student given 2 tests,

2nd student given 3 tests.

```
int[][] marks = {
    {80,90},
```

```
{70, 95, 80}  
};
```

Once user input all data, print following information.

1. Minimum marks of each student
2. Maximum marks of each student
3. Average marks of each student in class
4. Average marks of the class
5. Which student given maximum tests print just index. Assume, index represent student id

4. Stage a2 (access)

Assignment

Task 1:

Make GradeBook class with courseName and grades fields. Where courseName is a String and marks is 2-dimensional int array. Each row represent one student and each column of that row contains marks of different tests of that particular student.

Below array is just an example, it contains marks of 2 students, 1st student given 2 tests, 2nd student given 3 tests.

```
int[][] marks = {  
    {80,90},  
    {70, 95, 80}  
};
```

Get number of students from user input. Number of tests for each student may be different, so your program shall take number of tests from user input too, of each student. After fully initializing the GradeBook object, print following information.

1. Which student given maximum tests, print just index. Assume, index represent student id
2. Who got maximum average marks, print the row index only
3. Average marks of the class

4. Average marks of each student in class
 5. Minimum marks of each student
 6. Maximum marks of each student
-
7. Average, minimum and maximum marks of the student, of user given student ID, so get student ID from user input. (student ID is just an row index)

Write different methods in GradeBook class that perform above listed operations. Call that method from main method after initializing the GradeBook object with required data.

Statement Purpose:

Objective of this lab is to explore ArrayList data structure.

Activity Outcomes:

Students will be able to use effectively Arraylists and its method.

Instructor Note:

Students should have understanding about basic operations of Arrays.

1. Stage J (Journey)

Introduction

The Java ArrayList is a dynamic array-like data structure that can grow or shrink in size during the execution of a program as elements are added/deleted. An Array on the other hand, has a fixed size: once we declared it to be a particular size, that size cannot be changed. To use an ArrayList, you first have to import the class:

```
import java.util.ArrayList;
```

You can then create a new ArrayList object:

```
ArrayList<T> Test = new ArrayList<>();
```

The Java API has a list of all the methods provided by an ArrayList.

2. Stage a1 (apply)

Lab Activities

Activity 1

The following program shows a simple use of ArrayList. An array list is created, and then objects of type String are added to it. The list is then displayed. Some of the elements are removed and the list is displayed again.

Solution

```
package arraylistclassexample1;

import java.util.ArrayList;
public class ArrayListClassExample1 {
    public static void main(String[] args) {
        // TODO code application logic here
        ArrayList<String> list=new ArrayList<>();
        //list.
        list.add("Tooba");
        list.add("Maryam");
        list.add("Maryam");
        list.add("Tooba");
        System.out.println("Total size of list:"+list.size());
        display(list);
        System.out.println(list.lastIndexOf("Tooba"));
    }
}
```



```

list.add(0,"Ali");
System.out.println("Total size of list:"+list.size());
display(list);

list.remove("Tooba");
System.out.println("Total size of list:"+list.size());
display(list);

list.remove(1);
System.out.println("Total size of list:"+list.size());
display(list);

list.set(0, "Ayesha");
System.out.println("Total size of list:"+list.size());
display(list);

list.clear();
System.out.println("Total size of list:"+list.size());
display(list);
}
public static void display(ArrayList<String> a)
{
    for(String x:a)
    {
        System.out.println(x);
    }
    System.out.println();
}
}

```

3. Stage ▼ (verify)

Home Activities

Activity 1

- Create an arraylist of strings named student and apply given methods on array list.
 - Add name of students in arrayList
 - Ali
 - Ahmad
 - Umar

- Talha
- Create a static function named *display(ArrayList<String> students)* and call it after each of following steps and also print a blank line after each step.
- Add two more students at the end of students list using *students.add()*
 - 1)Tooba 2) Waleed
- Print the size of list.
- Add “Hamza” in the start of students.
- Add “Rizwan” as second element of list.
- Sort list of students
- Remove “Tooba” from list.
- Remove last element from list.

Hint: To remove last element use *(size()-1)*

- Display second element from list.
- Update name of first student. New name should be “Muhammad Waleed”
- Remove the first name.
- Remove all elements from list.

Activity 2

Write a program that uses an *ArrayList* of parameter type *Contact* to store a database of contacts. The *Contact* class should store the contact’s first and last name, phone number, and email address. Add appropriate accessor and mutator methods. Your database program should present a menu that allows the user to add a contact, display all contacts, search for a specific contact and display it, or search for a specific contact and give the user the option to delete it. The searches should find any contact where any instance variable contains a target search string. For example, if “elmore” is the search target, then any contact where the first name, last name, phone number, or email address contains “elmore” should be returned for display or deletion. Use the “for-each” loop to iterate through the *ArrayList*.

4. Stage a2 (access)

Assignment

Task 1:

Create a fully encapsulated class named *Student*.

Student class has three attributes name, id and CGPA. Create a constructor, which take two attributes i.e. id and name.

- Create test class with name *StudentTest*

In main class:

- Create three objects of Student named s1, s2 and s3 with following attributes
 - Ali,SP19-BSE-010
 - Ahmad", SP19-BSE-001
 - Talha, SP19-BSE-002
- Create a static function named *display*, call it after each of following steps, and print a blank line after each step. This method should display id, name and CGPA of student.

Hint: set CGPA using setCGPA(double cgpa) method

- Create an ArrayList named arraylist and add the above objects to arraylist
- Add another Student object at the start of ArrayList with the following attributes
 - Waleed, SP19-BSE-020
- Add another Student object at the end of ArrayList with the following attributes
 - Maha, SP19-BSE-002
- Remove second student from list using index.
- Remove first student from list using object reference.
- Print total number of students.
- Replace the student at index 0 with new student.
- Now display information of first student in list.

Statement Purpose:

The objective of this lab is to familiarize the students with various concepts and terminologies of inheritance using Java.

Activity Outcomes:

This lab teaches you the following topics:

- Declaration of the derived classes along with the way to access of base class members.
- Protected Access modifier and working with derived class constructors.

Instructor Note:

The student must know the procedure of creating class, method and objects.

1. Stage J (Journey)

Introduction

a. Inheritance

Inheritance is a way of creating a new class by starting with an existing class and adding new members. The new class can replace or extend the functionality of the existing class. The existing class is called the base class and the new class is called the derived class.

b. Protected Access Specifier

Protected members are directly accessible by derived classes but not by other users. A class member labeled **protected** is accessible to member functions of derived classes as well as to member functions of the same class.

c. Derived class constructor

Constructors are not inherited, even though they have public visibility. However, the super reference can be used within the child's constructor to call the parent's constructor. In that case, the call to parent's constructor must be the first statement.

2. Stage a1 (apply)

Lab Activities

Activity 1:

This example will explain the method to specify the derived class. It explains the syntax for writing the constructor of derived class.

Solution:

```
public class person {  
  
    protected String name ;  
    protected String id ;  
    protected int phone ;  
  
    public person() {  
        name = "NaginaNazar" ;  
        id = "sp14bcs039" ;  
        phone = 12345 ;  
    }  
}
```

```

        public person(String a , String b , int c) {
            name = a ;
            id = b ;
            phone = c ;
        }

        public void setName(String a){
            name = a ;
        }

        public void setId(String j){
            id = j ;
        }

        public void setPhone(int a) {
            phone = a ;
        }

        public String getName() {
            return name ;
        }

        public String getId() {
            return id ;
        }

        Public int getPhone() {
            return phone ;
        }

    }
    public class student extends person {
        private String rollNo ;
        private int marks ;

        public student() {
            super() ;
            rollNo = "sp14bcs039" ;
            marks = 345 ;
        }

        public student(String a , String b , int c , String
            d , int e){ super(a,b,c) ;
            rollNo = d ;
            marks = e ;
        }

        public void setRollNo(String a){
            rollNo = a ;
        }
    }

```

```

        public void setMarks(int a ){
            marks = a ;
        }
        public String getRollNo() {

            return rollNo ;

        }

        Public int getMarks() {
            return marks ;
        }

        public void display() {
            System.out.println("Name : " + name + "ID : " + id + "Phone : " + phone ) ;
            System.out.println("Roll # : " + rollNo + "\nMarks : " + marks) ;

        }

    }

    public class Runner
    {
        public static void main(String []args)
        {
            Student s = new Student ("s-09",Ahmed","xyz","sp16-bcs-98,50);
            s.display();
        }
    }

```

Activity 2:

This example demonstrates another scenario of inheritance. The super class can be extended by more than one class.

Solution:

```

public class Employee {

    protected String name;
    protected String phone;
    protected String address;
    protected int allowance;

    public Employee(String name, String phone, String address, int allowance)
    {
        this.name = name;
        this.phone = phone;
    }
}

```

```

        this.address = address;
        this.allowance = allowance;
    }
}
public class Regular extends Employee
{
    Private int basicPay;
    public Regular(String name, String phone, String address, int allowance, int basicPay)
    {
        super(name, phone, address, allowance);
        this.basicPay = basicPay;
    }
    public void Display(){
        System.out.println("Name: " + name + "Phone Number: " + phone + "Address: " +
        address
        + "Allowance: " + allowance + "Basic Pay: " + basicPay);    }
}
public class Adhoc extends Employee
{
    private int numberOfWorkingDays;
    private int wage;

    public Adhoc(String name, String phone, String address, int allowance,
    int
    numberOfWorkingDays, int wage)
    {
        super(name, phone, address, allowance);
        this.numberOfWorkingDays = numberOfWorkingDays;
        this.wage = wage;
    }
    public void Display()
    {
        System.out.println("Name: " + name + "Phone Number: " + phone + "Address: " +
        address
        + "Allowance: " + allowance + "Number Of Working Days:
        "
        +
        numberOfWorkingDays + "Wage: " + wage);
    }
}
public class Runner
{
    public static void main(String []args){
        Regular regularObj = new
        Regular("Ahmed", "090078601", "Islamabad", 15000, 60000);
        regularObj.Display();
        Adhoc adhocObj = new Adhoc("Ali", "03333333333", "Rawalpindi", 500, 23, 1500);
        adhocObj.Display();
    }
}

```


3. Stage **v** (verify)

Home Activities

Activity 2

Make a class **Circle** that has two private attribute **color** and **radius**. Create a fully parameterized constructor. This class has a method *calculateArea()*, which calculates area of circle.

Make a class **Cylinder** that has one private attribute height. **Here Cylinder is a Circle.** Create a fully parameterized constructor. This class has a method *calculateVolume()*, which calculates volume of cylinder.

In test class, create an object of cylinder. Ask user to input value of color, radius and height. Print color, radius, and area of circle and volume of cylinder.

Put all classes in com.oop.lab.inheritance package.

Expected Output:

```
run:
Enter color:Red
Enter radius:2
Enter heighth:3
Color is Red
Radius is 2.0
Heighth is 3.0
Area of circle is 12.56
Volume of cylinder is 37.68
BUILD SUCCESSFUL (total time: 34 seconds)
```

4. Stage **a2** (access)

Assignment

Task 1:

Make a class **Address** which has two private attribute city and country and one fully parametrized constructor.

Make a class **Person** that has two private attribute name and address. Attribute address should be of type Address. This class has fully parameterized constructor.

Make another class **Student**. Class Student is inherited from class Person. This class has three private attributes program, year and fee and one fully parametrized constructor.

Make a class **Staff** that is inherited from class Person. This class has one attribute that is pay and has one fully parameterized constructor.

In test class, create one object for Student and one for Staff. Ask user to input all information of Student and Staff and print values of all attributes.

All attributes are private.

Expected Output:

```
run:
Enter Student Information
Enter name:Ali
Enter city:Lahore
Enter country:Pakistan
Enter program:BSCS
Enter year:2019
Enter fee:85000
Enter Staff Information
Enter name:Maryam
Enter city:Islamabad
Enter country:Pakistan
Enter pay:80000

Student information is given below:
Name: Ali
Address: Lahore Pakistan
Program: BSCS
Year: 2019
Fee: 85000

Staff information is given below:
Name: Maryam
Address: Islamabad Pakistan
Pay: 80000
```

Task 2:

This task is taken from book exercise:

1. (Employee Hierarchy) In this chapter (Chapter 9), you studied an inheritance hierarchy in which class BasePlusCommissionEmployee inherited from class CommissionEmployee. However, not all types of employees are CommissionEmployees. In this exercise, you'll create a more general Employee

superclass that factors out the attributes and behaviors in class `CommissionEmployee` that are common to all `Employees`. The common attributes and behaviors for all `Employees` are `firstName`, `lastName`, `socialSecurityNumber`, `getFirstName`, `getLastName`, `getSocialSecurityNumber` and a portion of method `toString`. Create a new superclass `Employee` that contains these instance variables and methods and a constructor. Next, rewrite class `CommissionEmployee` from Section 9.4.5 as a subclass of `Employee`. Class `CommissionEmployee` should contain only the instance variables and methods that are not declared in superclass `Employee`. Class `CommissionEmployee`'s constructor should invoke class `Employee`'s constructor and `CommissionEmployee`'s `toString` method should invoke `Employee`'s `toString` method.

Once you've completed these modifications, run the `CommissionEmployeeTest` and `BasePlusCommissionEmployeeTest` apps using these new classes to ensure that the apps still display the same results for a `CommissionEmployee` object and `BasePlusCommissionEmployee` object, respectively.

2. (Creating a New Subclass of `Employee`) Other types of `Employees` might include `SalariedEmployees` who get paid a fixed weekly salary, `PieceWorker` who get paid by the number of pieces they produce or `HourlyEmployees` who get paid an hourly wage with time-and-a-half—1.5 times the hourly wage—for hours worked over 40 hours.

Create class `HourlyEmployee` that inherits from class `Employee` and has instance variable `hours` (a double) that represents the hours worked, instance variable `wage` (a double) that represents the wages per hour, a constructor that takes as arguments a first name, a last name, a social security number, an hourly wage and the number of hours worked, set and get methods for manipulating the hours and wage, an earnings method to calculate an `HourlyEmployee`'s earnings based on the hours worked and a `toString` method that returns the `HourlyEmployee`'s String representation. Method `setWage` should ensure that wage is nonnegative, and `setHours` should ensure that the value of hours is between 0 and 168 (the total number of hours in a week). Use class `HourlyEmployee` in a test program that's similar to the one in Fig. 9.5.

3. Update your code to meet following new requirement. If an employee makes high commission, company want to give him/her bonus as per following formula:

Commission		Bonus
10000-20000		5000
20000-50000		10000
50000+		20000

Task 3:

Modify the CommissionEmployee class earnings method in such a way that bonus shall be added for all type of employees who are belong to CommissionEmployee or its any sub-type.

Imagine a publishing company that markets both book and audio-cassette versions of its works. Create a class publication that stores the title and price of a publication. From this class derive two classes:

- i. book, which adds a page count and
- ii. tape, which adds a playing time in minutes.

Each of these three classes should have set() and get() functions and a display() function to display its data. Write a main() program to test the book and tape class by creating instances of them, asking the user to fill in their data and then displaying the data with display().

Statement Purpose:

The objective of this lab is to familiarize the students with various concepts and terminologies of method overriding.

Activity Outcomes:

This lab teaches you method overriding where a base class method version is redefined in the child class with exact method signatures.

Instructor Note:

The Students should know the implementation of inheritance.

1. Stage J (Journey)

Introduction

Method Overriding

The definition of an inherited method can be changed in the definition of a derived class so that it has a meaning in the derived class that is different from what it is in the base class. This is called overriding the definition of the inherited method.

For example, the methods `toString` and `equals` are overridden (redefined) in the definition of the derived class `HourlyEmployee`. They are also overridden in the class `SalariedEmployee`. To override a method definition, simply give the new definition of the method in the class definition, just as you would with a method that is added in the derived class.

In a derived class, you can override (change) the definition of a method from the base class. As a general rule, when overriding a method definition, you may not change the type returned by the method, and you may not change a void method to a method that returns a value, nor a method that returns a value to a void method. The one exception to this rule is if the returned type is a class type, then you may change the returned type to that of any descendent class of the returned type. For example, if a function returns the type `Employee`, when you override the function definition in a derived class, you may change the returned type to `HourlyEmployee`, `SalariedEmployee`, or any other descendent class of the class `Employee`. This sort of changed return type is known as a covariant return type and is new in Java version 5.0; it was not allowed in earlier versions of Java.

2. Stage a1 (apply)

Lab Activities

Activity 1:

The following activity demonstrates the creation of overridden methods.

Solution:

```
class A
{
    int i, j;

    A(int a, int b) {
```

```

        i = a;
        j = b;
    }
    // display i and j
    void
    show()
    {
        System.out.println("i and j: " + i + " " + j);
    }
}
class B extends A
{
    int k;
    B(int a, int b, int c) {
        super(a, b);
        k = c;
    }
    // display k – this overrides
    show() in A void show() {
        System.out.println("k: " + k);
    }
}
Public class OverrideRunner
{
    public static void main(String args[])
    {
        B subOb = new B(1, 2, 3);
        subOb.show(); // this calls show() in B
    }
}

```

Output:

The output produced by this program is shown here:
k: 3

Activity 2:

The following activity explains the use of overriding for customizing the method of super class. The classes below include a CommisionEmployee class that has attributes of firstname, lastName, SSN, grossSales, CommisionRate. It has a constructor to initialize, set and get functions, and a function to display data members. The other class BasePlusCommisionEmployee is inherited from CommisionEmployee. It has additional attributes of Salary. It also has set and get functions and display function. The Earning method is overridden in this example.

Solution:

```
public class commissionEmployee
{
    protected String FirstName;
    protected String LastName;
    protected String SSN;
    protected double grossSales;
    protected double commonRate;
    public commissionEmployee()
    {
        FirstName="Nagina";
        LastName="Nazar";
        SSN="S003";
        grossSales=1234.1;
        commonRate=12.5;
    }
    Public commissionEmployee (String a,Stringe,String b, double c, double
    d){ FirstName=a;
        LastName=e;
        SSN=b;
        grossSales=c;
        commonRate=d;
    }
    public void setFN(String a){
        FirstName=a;
    }
    public void setLN(String e){
        LastName=e;
    }
    public void setSSN(String b){
        SSN=b;
    }
    public void setGS(double c){
        grossSales=c;
    }
    public void setCR(double d){
        commonRate=d;
    }
    public String getFN(){
        returnFirstName;
    }

    public String getSSN(){
        return SSN;
    }
    public double getGS(){
        returngrossSales;
    }
    public double getCR(){
```



```

        return commonRate;
    }
    public double earnings(){
        return grossSales*commonRate;
    }
    public void display(){
        System.out.println("first name:"+FirstName+",last
name:"+LastName+" ,SSN:"+SSN+" Gross Sale:"+grossSales+" and
commonRate:"+commonRate);
    }
}

```

```

public class BasePlusCommEmployee extends
commissionEmployee{ private double salary;

    BasePlusCommEmployee(){
        salary=48000;
    }
    BasePlusCommEmployee(String A,String E,String B, double C, double D,
        double S){ super(A,E,B,C,D);
        salary=S;
    }
    //overridden method
    public double earnings(){
        return super.earnings()+salary;
    }
    public void display(){
        super.display();
        System.out.println("Salary : "+salary);
    }
}

```

```

Public class OverrideRunner
{
    public static void main(String args[])
    {
        BasePlusCommEmployee b = new BasePlusCommEmployee("ali", "ahmed",
        "25-kkn", 100, 5.2, 25000);
        double earn = b.earnings();
        System.out.println("Earning of employee is " + earn);
    }
}

```

3. Stage v (verify)

Home Activities

Activity 1

Make a class **Shape** that has one private attribute **color**. Create a fully parameterized constructor. This class has two public methods *calculateArea()* and *calculatePerimeter()*. Both methods return 0.

Make a class **Circle** that has one private attribute **radius**. Class **Circle** is inherited from class **Shape**. Create a fully parameterized constructor.

Make a class **Rectangle** that extends **Shape**, and it has two private attributes **width** and **length**. Create a fully parameterized constructor.

All sub classes override the *calculateArea()* and *calculatePerimeter()*.

Declare the method of Shape class “calculateArea()” as final. Check the error. Declare the Shape class as final. Check the error.

In test class create ask user to input all values and display area and perimeter for **Circle** and **Rectangle**.

Put all classes in `com.oop.lab.overriding` package.

Expected Output:

```
run:
Enter color of circle:Black
Enter radius:3.5
Enter color of rectangle:Red
Enter length:5
Enter width:2

Color of Circle:Black
Area of Circle:38.465
Perimeter of Circle:21.98

Color of Rectangle:Red
Area of Rectangle:10.0
Perimeter of Rectangle:14.0
BUILD SUCCESSFUL (total time: 37 seconds)
```

Activity 2

Make a class **Circle** that has two private attributes **color** and **radius**. Create a fully parameterized constructor. This class has a method *calculateArea()*, which calculates

area of circle. This class is inherited by **Cylinder** that has one private attribute height. Create a fully parameterized constructor. This class overrides *calculateArea()* **method**. In test class, call calculateArea() method of Cylinder.

$$\text{Area of Cylinder} = 2(\pi r^2 + \pi rh)$$

Hint: πr^2 = Area of Circle

Expected Output:

4. Stage a2 (access)

Assignment

Task 1:

Make a class **CommissionEmployee** that is inherited by **BasePlusCommissionEmployee**. Class **CommissionEmployee** has four private attribute firstName, LastName, grossSales and commissionRate. Create a fully parameterized constructor. In this class, you have to check that grossSales must be greater than zero and commissionRate must be greater than zero and less than 1. This class has 2 methods, *earnings()* and *toString()*. You can calculate earnings of employee using this formula: $\text{commissionRate} * \text{grossSales}$.

Make a class **BasePlusCommissionEmployee** that has one private attribute baseSalary. Create a fully parameterized constructor. In this class, you have to check that baseSalary must be greater than 500. This class overrides both methods *earnings()* and *toString()*. You can calculate earnings of employee using this formula: $\text{commissionRate} * \text{grossSales} * \text{baseSalary}$.

In the test class, get the firstName, lastName, grossSale, commissionRate and baseSalary from user and call *toString()* and *earnings()* methods.

Statement Purpose:

Objective of this lab is to make students understand the polymorphism, up casting and down casting.

Activity Outcomes:

The student will understand the basic paradigm of OOP i.e. Polymorphism.

Instructor Note:

The Students should have implemented inheritance and method overriding.

1. Stage J (Journey)

Introduction

Generally, the ability to appear in many forms. In object-oriented programming, *polymorphism* refers to a programming language's ability to process objects differently depending on their data type or class. More specifically, it is the ability to redefine *methods* for *derived classes*. For example, given a base class *shape*, polymorphism enables the programmer to define different *area* methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the *area* method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language (OOPL).

Polymorphism could be static and dynamic both. Method Overloading is static polymorphism while, Method overriding is dynamic polymorphism.

- **Overloading** in simple words means more than one method having the same method name that behaves differently based on the arguments passed while calling the method. This called static because, which method to be invoked is decided at the time of compilation
- **Overriding** means a derived class is implementing a method of its super class. The call to overridden method is resolved at runtime, thus called runtime polymorphism

Up casting and down casting:

- ♦ You cannot automatically make reference variable of subclass type point to object of its superclass
- ♦ Suppose that supRef is a reference variable of a superclass type and supRef points to an object of its subclass:
 - ♦ Can use a cast operator on supRef and make a reference variable of the subclass point to the object
 - ♦ If supRef does not point to a subclass object and you use a cast operator on supRef to make a reference variable of the subclass point to the object, then Java will throw a `ClassCastException`—indicating that the class cast is not allowed

Upcasting and Downcasting

Upcasting occurs when an object of one type is assigned to a variable declared with a supertype of the object. No explicit casting is needed:

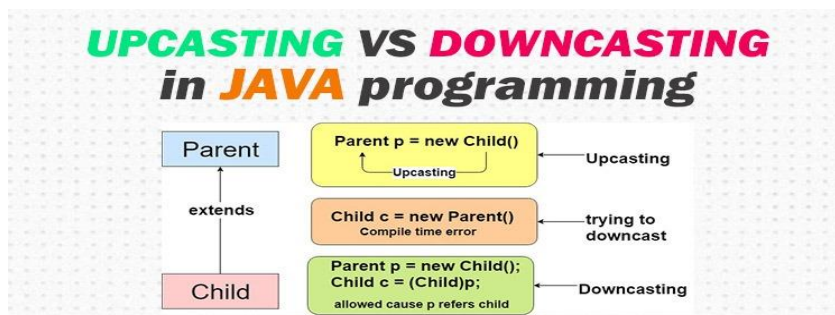
```
Polyhedron p = new Cube(5);
```

Downcasting, however, requires an explicit cast:

```
Cube c = (Cube) p;
```

CSE 341 -- S. Tanimoto

Java 3 - 15



- ♦ Operator instanceof: determines whether a reference variable that points to an object is of a particular class type
- ♦ This expression evaluates to true if p points to an object of the class BoxShape; otherwise it evaluates to false:

```
p instanceof BoxShape
```

2. Stage a1 (apply)

Lab Activities

Activity 2

Following example, demonstrate polymorphism in java.

Solution

```

package poly;
public class Animal {

    public String name;
    public Animal(){
        name = "MyPet";
    }

    public void eat(){
        System.out.println("Animal is eating food!!");
    }
}

package poly;

public class Dog extends Animal{
    public void eat(){
        System.out.println("Dog is eating bone!!");
    }
}

package poly;

public class Cow extends Animal{
    public void eat(){
        System.out.println("Cow is eating grass!!");
    }
}

package poly;

public class Poly {

    public static void main(String[] args) {

        Animal a1 = new Animal();
        Animal a2 = new Dog();//upcasting
        Animal a3 = new Cow(); //upcasting

        System.out.println("Name is: "+a2.name);
    }
}

```

```
        a1.eat();
        a2.eat();
        a3.eat();
    }

}
```

3. Stage v (verify)

Home Activities

Activity 1

Make a class Faculty that is inherited by 2 classes PermanentFaculty and VisitingFaculty.

Class Faculty has two attributes id and name, fully parametrized constructor and calculateSalary() method.

Class PermanentFaculty has one attribute salary, fully parametrized constructor and overrides calculateSalary() method.

Class VisitingFaculty() has two attributes hours,salaryPerHr, fully parametrized constructor and overrides calculateSalary() method.

In PermanantFaculty class, *calculateSalry* method returns basic salary. In VisitingFaculty class, *calculateSalry* method returns hours*salaryPerHr.

In Test class create an array of Faculty (n elements) and ask user which object data to enter, 1 for PermanentFaculty, 2 for Visiting. Input data for appropriate object and save in the array.

After n objects are saved, show salary of all objects through polymorphic processing. Now update salary of permanantFaculty by 10%. (HINT: You will have to first check for required Faculty using the instanceof operator, then, you will have to downcast to set appropriate value of the object)

Expected Output:


```

-----
Enter number of Employee:2
Enter choice:1
Enter employee id:123
Enter employee name:Ali
Enter employee salary:90000
Enter choice:2
Enter employee id:567
Enter employee name:Tooba
Enter employee hour:32
Enter salary per hr:1500
Salary of Permanant Faculty is: 90000.0
Salary of visiting Faculty is: 48000.0

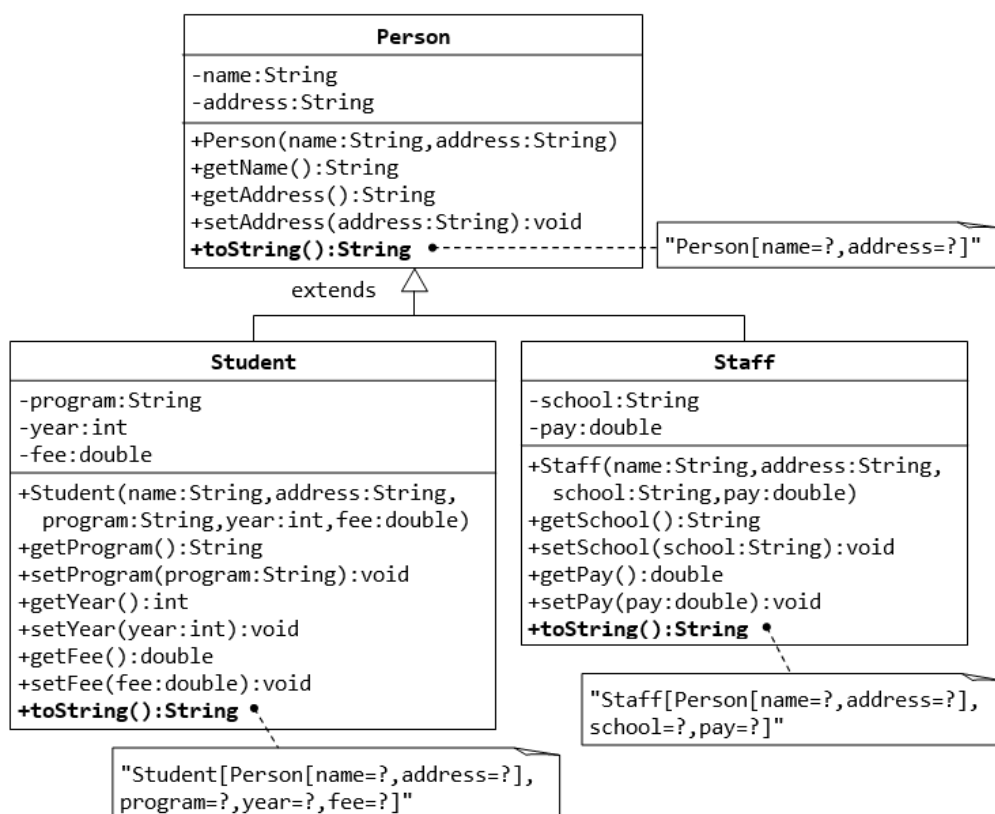
Increased salary of Permanant faculty is:99000.0

```

4. Stage a2 (access)

Assignment

Task 1:



In main class, create an ArrayList of Person (3 elements). Ask user which object data to enter, 1 for staff, 2 for student. Input data for appropriate object and save in the

ArrayList. After 3 objects are saved, show all objects (using the toString method) through polymorphic processing.

Expected Output:

```
Enter number of person:3
Enter choice 1 for staff and 2 for student:1
Enter employee name:Ali
Enter employee address:Lahore
Enter university name:COMSATS
Enter employee salary:80000
Enter choice 1 for staff and 2 for student:1
Enter employee name:Tooba
Enter employee address:Islamabad
Enter university name:Ripah
Enter employee salary:70000
Enter choice 1 for staff and 2 for student:2
Enter Student name:Rimsha
Enter Student address:Lahore
Enter program:BSCS
Enter year:2019
Enter fee:85000
Person{name=Ali, address=Lahore}Staff{school=COMSATS, pay=80000.0}
Person{name=Tooba, address=Islamabad}Staff{school=Ripah, pay=70000.0}
Person{name=Rimsha, address=Lahore}Student{program=BSCS, year=2019, fee=85000.0}
```

Statement Purpose:

The objective of this lab is to familiarize the students with abstract classes.

Activity Outcomes:

This lab teaches you the following topics:

- Abstract classes along with the access of base class members.

Instructor Note:

The Students should know the implementation of inheritance.

1. Stage J (Journey)

Introduction

A class that has at least one abstract method is called an abstract class and, in Java, must have the modifier abstract added to the class heading. An abstract class can have any number of abstract methods. In addition, it can have, and typically does have, other regular (fully defined) methods. If a derived class of an abstract class does not give full definitions to all the abstract methods, or if the derived class adds an abstract method, then the derived class is also an abstract class and must include the modifier abstract in its heading.

In contrast with the term abstract class, a class with no abstract methods is called a concrete class.

2. Stage a1 (apply)

Lab Activities

Activity 3:

Abstract Class

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

Output:

B's implementation of Call me.

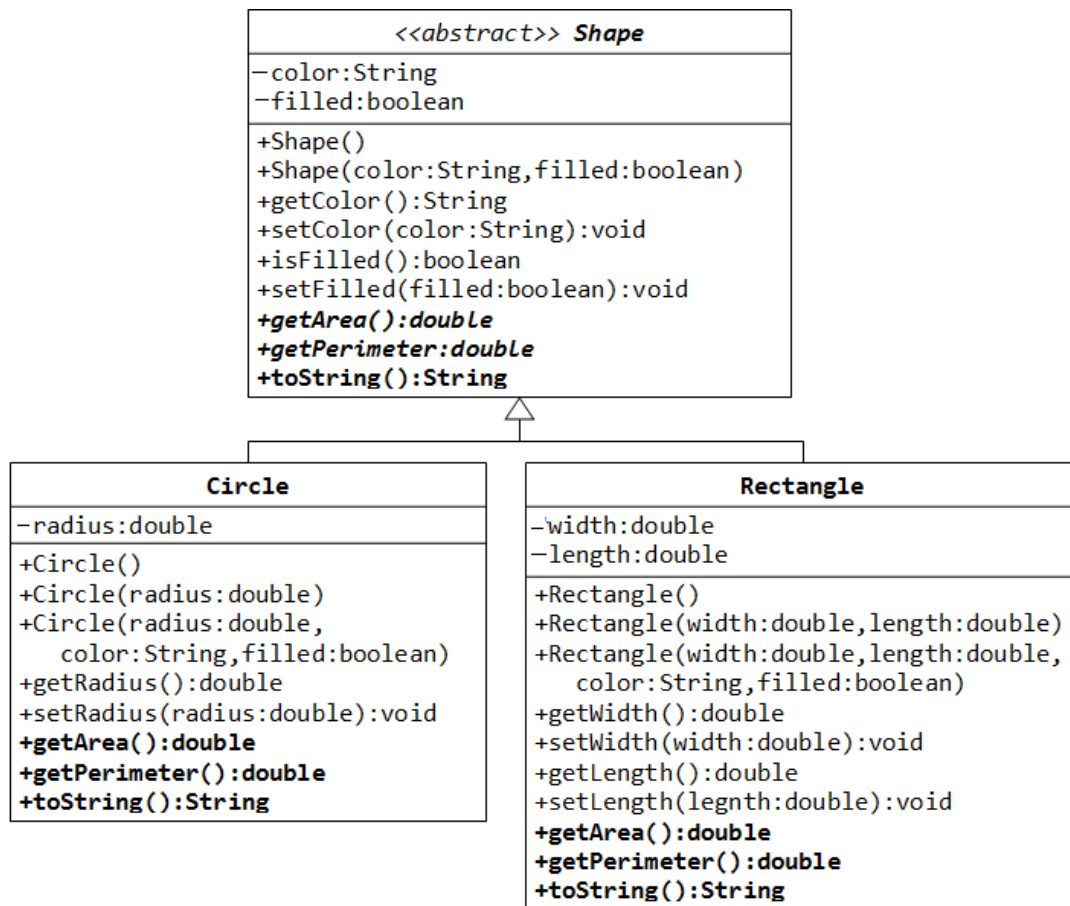
This is a concrete method

3. Stage ▼ (verify)

Home Activities

Activity 1

Write code using following UML diagram. Starter code of **shape, circle and rectangle class** is uploaded on google classroom. Download the code from Google Classroom and update it.



1. Create an array of shapes of 4 elements.
2. Initialize array with elements of different sub-classes (rectangle, circle).
3. Show the user a menu, which shape's data he wants to change (circle:1, or rectangle:2).
 - Input shape data from user (length, width in case of rectangle, and radius in case of circle)
 - Now inside a loop, set the data of all shapes of input type to input values. E.g. if the user enters 1 for shape (i.e. circle), then 5 for radius of the circle, then the radius of all circle objects in the array should be set to 5. (HINT: You will have to first check for required shape using the instanceof operator, then, you will have to downcast to set appropriate value of the object)
 - Check the desired shape.
 - Downcast the shape in desired shape.
 - Change values according to user input.
 - Set back the array element.
4. Create another static method called `shapesSummary(Shape [] s)` which displays shapes area, perimeter and prints the shapes using the `toString` method.
5. In the end, call the `shapesSummary` method to print the data of all the shapes.

Expected Output:

```
Enter choice, 1 for Circle, 2 for Rectangle:1
Enter radius:4.5
Shape{color=Blue, filled=true}Circle{radius=4.5}
Area:63.585
Perimeter:28.26
Shape{color=Red, filled=false}Circle{radius=4.5}
Area:63.585
Perimeter:28.26
Shape{color=yellow, filled=false}Recatngle{width=2.0, length=4.0}
Area:8.0
Perimeter:16.0
Shape{color=Green, filled=true}Recatngle{width=3.0, length=5.0}
Area:15.0
Perimeter:30.0
```

4. Stage a2 (access)

Assignment

Task 1:

Create an Abstract class Student that contains a method take exam, implement the method in the child classes PhdStudent and GradStudent in which PhdStudent takes exam by giving his final defense presentation while the graduate student gives a written paper.

Statement Purpose:

Objective of this lab is to learn how to define Interface and implement Interfaces. Students will also learn how to implement multiple interfaces in one class.

Activity Outcomes:

This lab teaches you the following topics:

- Interface class.
- Implementation of Interface classes and its methods.
- Implementation of multiple interfaces in class.
- Use of interfaces with abstract classes

Instructor Note:

The Students should know the implementation of inheritance and abstract classes.

1. Stage J (Journey)

Introduction

An interface is a type that groups together a number of different classes that all include method definitions for a common set of method headings.

Java interface specifies a set of methods that any class that implements the interface must have. An interface is itself a type, which allows you to define methods with parameters of an interface type and then have the code apply to all classes that implement the interface. One way to view an interface is as an extreme form of an abstract class. However, as you will see, an interface allows you to do more than an abstract class allows you to do. Interfaces are

Java's way of approximating multiple inheritance. You cannot have multiple base classes in Java, but interfaces allow you to approximate the power of multiple base classes.

Syntax:

How to define interface

```
public interface Ordered {  
  
    public boolean precedes(Object other);  
  
}
```

How to implement interface and its method

```
public class Class_name implements Interface_Name{  
  
    public boolean precedes(Object other){  
  
    }  
  
}  
  
public class implements SomeInterface, AnotherInterface{ }
```

Abstract Classes Implementing Interfaces

A concrete class (that is, a regular class) must give definitions for all the method headings given in an interface in order to implement the interface. However, you can define an abstract class that implements an interface but gives only definitions for some of the method headings given in the interface. The method headings given in the interface that are not given definitions are made into abstract methods.

Derived Interfaces

You can derive an interface from a base interface. This is often called extending the interface.

The details are similar to deriving a class.

The Comparable Interface

The Comparable interface is in the java.lang package and so is automatically available to your program. The Comparable interface has only the following method heading that must be given a definition for a class to implement the Comparable interface:

```
public int compareTo(Object other);
```

The method compareTo should return a negative number if the calling object “comes before” the parameter other, a zero if the calling object “equals” the parameter other, and a positive number if the calling object “comes after” the parameter other. The “comes before” ordering

that underlies compareTo should be a total ordering. Most normal ordering, such as less-than ordering on numbers and lexicographic ordering on strings, is total ordering.

Defined Constants in Interfaces

The designers of Java often used the interface mechanism to take care of a number of miscellaneous details that do not really fit the spirit of what an interface is supposed to be. One example of this is the use of an interface as a way to name a group of defined constants. An interface can contain defined constants as well as method headings, or instead of method headings. When a method implements the interface, it automatically gets the defined constants. For example, the following interface defines constants for months:

```
public interface MonthNumbers {  
  
    public static final int JANUARY = 1,    FEBRUARY = 2, MARCH = 3, APRIL =  
    4,  
    MAY = 5, JUNE = 6, JULY = 7, AUGUST = 8, SEPTEMBER = 9, OCTOBER  
    = 10, NOVEMBER = 11, DECEMBER = 12; }  

```

Any class that implements the MonthNumbers interface will automatically have the 12 constants defined in the MonthNumbers interface. For example, consider the following toy class:

```
public class DemoMonthNumbers implements MonthNumbers {  
  
    public static void main(String[] args)    {  
  
        System.out.println( "The number for January is " + JANUARY);    } }  

```

2. Stage a1 (apply)

Lab Activities

Activity 1:

Declare a Interface, RegisterForExams that contains single method register, implements the interface in two different classes (a) Student (b) Employee. Write down a Test Application that contains at least a function that takes Interface type Parameter

Solution:

```
public interface RegisterForExams {
    public void register();
}

public class InterfaceTestClass {

    public InterfaceTestClass(RegisterForExams as)
    {
        as.register();
    }
}

public class EmployeeTask implements RegisterForExams{

    private String name;
    private String date;
    private int salary;

    public EmployeeTask()
    {
        name = null;
        date = null;
        salary = 0;
    }

    public EmployeeTask(String name,String date,int salary)
    {
        this.name = name;
        this.date = date;
        this.salary = salary;
    }

    @Override
    public void register() {
        // TODO Auto-generated method stub
        System.out.println("Name " + name + "\nsalary " + salary + "\nEmployee reistered on date " + date);
    }
}
```

```

}
public class StudentTask implements
    RegisterForExams{ private
        String name;
        private int age;
        private double gpa;

        public StudentTask()
        {
            name = null;
            age = 0;
            gpa = 0;
        }
        public StudentTask(String name,int age,double gpa)
        {
            this.name = name;
            this.age = age;
            this.gpa = gpa;
        }

    @Override
    public void register() {
        // TODO Auto-generated method stub
        System.out.println("Student name " + name + " gpa " + gpa);
    }
}

```

Activity 2:

This example show how to implement interface to a class

`public class OrderedHourlyEmployee extends HourlyEmployee implements Ordered{`

```

    public boolean precedes(Object other) {
        if (other == null)

            return false;
            else if (!(other instanceof OrderedHourlyEmployee))
                return false;

            else {

                OrderedHourlyEmployee otherOrderedHourlyEmployee =
                    (OrderedHourlyEmployee)other; return (getPay() <
                        otherOrderedHourlyEmployee.getPay());
            }
        }
        public boolean follows(Object other)
        {
            if (other == null)
                return false;
            else if (!(other instanceof OrderedHourlyEmployee))

```

```

        return false;
    else {
        OrderedHourlyEmployee otherOrderedHourlyEmployee =
            (OrderedHourlyEmployee)other; return (otherOrderedHourlyEmployee.precedes(
                this));
    } }
}

```

Activity 3:

An Example that shows How to create your own interface and implement it in abstract class

```

interface I1 {

    void methodI1(); // public static by default

}

interface I2 extends I1 {

    void methodI2(); // public static by default

}

class A1 {
    public String methodA1() {

        String strA1 = "I am in methodC1 of class A1";

        return strA1;

    }

    public String toString() {

        return "toString() method of class A1";

    }

}

class B1 extends A1 implements I2 {

    public void methodI1() {

        System.out.println("I am in methodI1 of class B1");

    }

    public void methodI2() {

        System.out.println("I am in methodI2 of class B1");

    }

}

class C1 implements I2 {

```

```

        public void methodI1() {
            System.out.println("I am in methodI1 of class C1");
        }
        public void methodI2() {
            System.out.println("I am in methodI2 of class C1");
        }
    }

// Note that the class is declared as abstract as it does not
// satisfy the interface contract
    abstract class D1 implements I2 {
        public void methodI1() {
        }
        // This class does not implement methodI2() hence declared abstract.
    }

    public class InterFaceEx {
    public static void main(String[] args) {
        I1 i1 = new B1();
        i1.methodI1(); // OK as methodI1 is present in B1
// i1.methodI2(); Compilation error as methodI2 not present in I1
// Casting to convert the type of the reference from type I1 to type I2 ((I2)
i1).methodI2();
        I2 i2 = new B1();
        i2.methodI1(); // OK
        i2.methodI2(); // OK
// Does not Compile as methodA1() not present in interface reference I1
// String var = i1.methodA1();
// Hence I1 requires a cast to invoke methodA1

        String var2 = ((A1) i1).methodA1();
        System.out.println("var2 : " + var2);
        String var3 = ((B1) i1).methodA1();
        System.out.println("var3 : " + var3);
        String var4 = i1.toString();
        System.out.println("var4 : " + var4);
        String var5 = i2.toString();
        System.out.println("var5 : " + var5);
        I1 i3 = new C1();
        String var6 = i3.toString();
        System.out.println("var6 : " + var6); // It prints the Object toString() method
        Object o1 = new B1();
// o1.methodI1(); does not compile as Object class does not define
// methodI1()
// To solve the problem we need to downcast o1 reference. We can do it
// in the following 4 ways
        ((I1) o1).methodI1(); // 1
        ((I2) o1).methodI1(); // 2
        ((B1) o1).methodI1(); // 3
        /*
        *
        * B1 does not have any relationship with C1 except they are "siblings".

```

```

* Well, you can't cast siblings into one another.
    *
    */
    // ((C1)o1).methodI1(); Produces a ClassCastException
}
}

```

3. Stage ▼ (verify)

Home Activities

Activity 1:

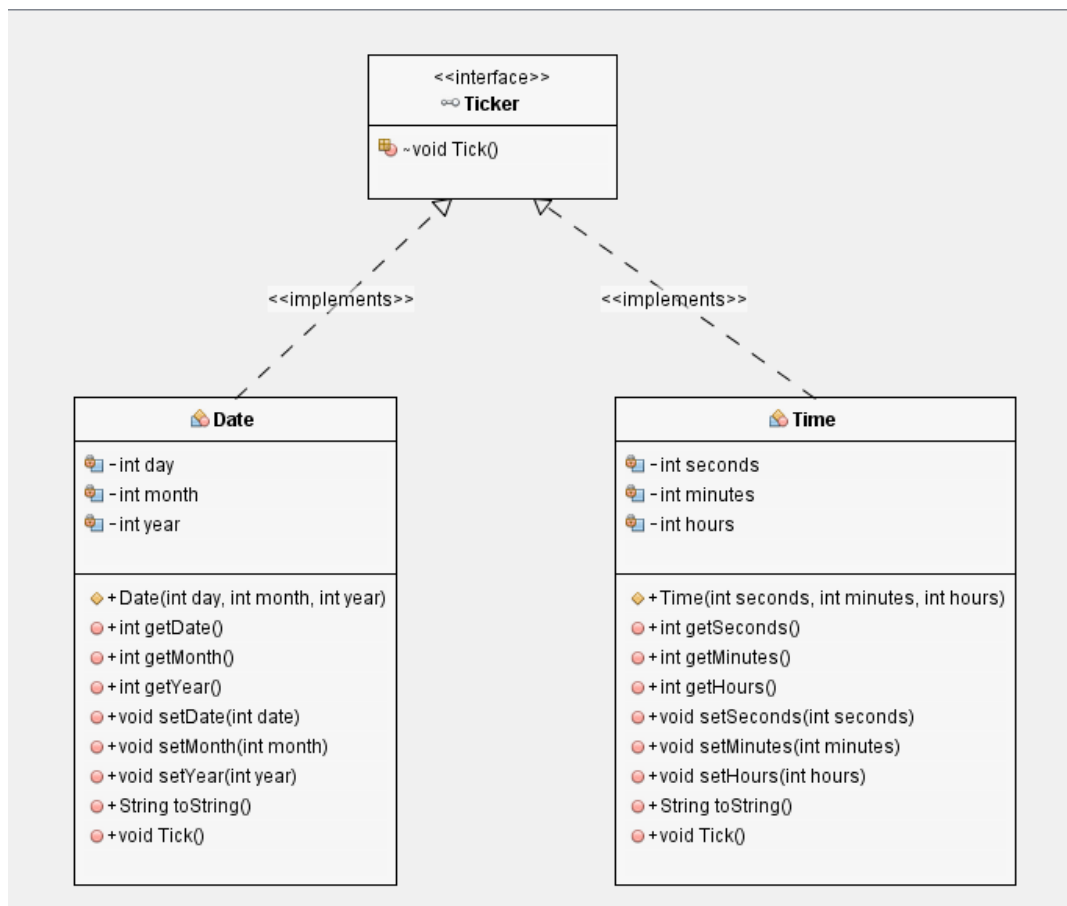
Create an interface EnhancedShape that extends Shape and also requires a method public double perimeter() to be implemented by any class that uses the interface.

Activity 2:

Write down a test application that implements EnhancedShape interface, call the interface methods to verify the functionality of implemented methods.

Activity 3:

Convert following UML diagram to java code.



Test Class:

Tick method in Time class

Increment in seconds and if second is equal to 60 rest it 0 and increment in minutes. If minutes is equal to 60 rest minutes to 0 and increment in hours. If hours is equal to 24 rest it to 0.

Tick method in Date class

Increment in days, if day is equal to 30 rest it 1 and increment in month. If month is equal to 12 rest month to 1 and increment in year.

Check expected output and create object of date and time class accordingly and call `toString()` for both.

Now set data and time and call `toString()`.

Now create polymorphic reference of **Ticker** interface with name `dateTime` and call `tick` method for date and time and then call `toString()`.


```

run:
Date{date=16, month=4, year=2019}
Time{seconds=1, minutes=2, hours=3}
Date{date=29, month=11, year=2019}
Time{seconds=59, minutes=59, hours=23}
Date{date=1, month=1, year=2020}
Time{seconds=0, minutes=0, hours=0}
BUILD SUCCESSFUL (total time: 0 seconds)

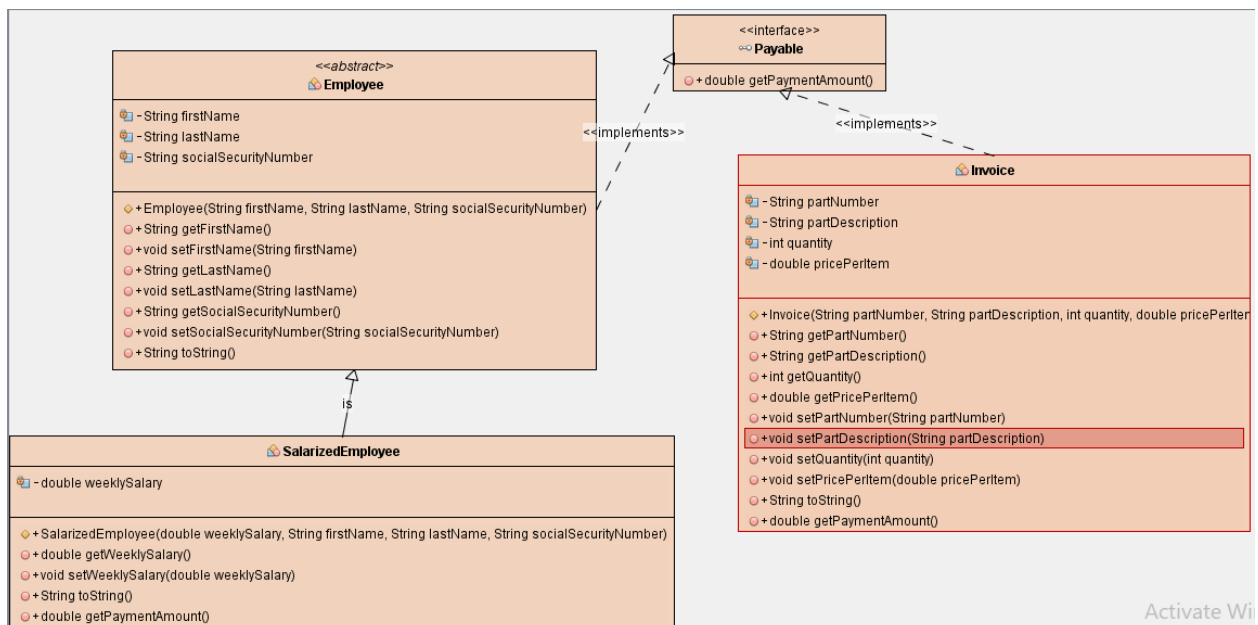
```

4. Stage a2 (access)

Assignment

Task 1:

Convert following UML diagram to java code.



getPaymentAmount() of **Invoice** class returns product of quantity and PricePerItem and *getPaymentAmount()* of **SalariedEmployee** class returns weeklySalary.

Test Class:

Create an array of Payable (4 elements). Initialize array with elements of **Invoice** class and **SalariedEmployee**. Now call *toString()* and *getPaymentAmount()*.

```
run:
Invoice partNumber=123, partDescription=seat, quantity=2, pricePerItem=375.0 PayMent Due750.0
Invoice partNumber=456, partDescription=tire, quantity=4, pricePerItem=79.95 PayMent Due319.8
SalariedEmployee firstName=Muhammad, lastName=Ali, socialSecurityNumber=111-11-1111 weeklySalary=800.0 PayMent Due800.0
SalariedEmployee firstName=Muhammad, lastName=Talha, socialSecurityNumber=888-88-8888 weeklySalary=1200.0 PayMent Due1200.0
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Task 2

Define an interface named Shape with a single method named area that calculates the area of the geometric shape: public double area();

Task 3

Implement the Shape interface for Rectangle, Circle and Triangle class.

Task 4

Implement a class CalculateAreas that has a function that takes shape type array of objects and builds an array of (double values) values for each corresponding shapes.

Statement Purpose:

1. Understanding exception-throwing methods.
2. Using try-catch to handle exceptions.
3. Understanding and writing custom exceptions.

Activity Outcomes:

This lab teaches you the following topics:

- Exception handling
- (try, catch, finally, throw and throws) to handle exceptions.

Instructor Note:

Student should know exception hierarchy and implementation of classes and inheritance.

1. Stage J (Journey)

Introduction

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and *thrown* in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is *caught* and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method. Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.

Exception-Throwing Methods

Runtime errors appear in Java as exceptions, exception is a special type of classes that could be thrown to indicate a runtime error and provide additional data about that error. If a method is declared to throw an exception, any other method calls it should deal with this exception by throwing it (if it appears) or handle it.

try-catch Exception Handling

Another technique for handling runtime errors is to use try-catch block to handle different types of exceptions and take appropriate action instead of throwing them to the user. The format of try-catch block is the following.

```
try{
    //Statement(s) that may throw exceptions
} catch(Exception e){
    //Statement(s) to handle exception.
}
```

When we are expecting more than one exception, we can use several catch blocks, each one for different type of exceptions.

Finally

You can attach a finally-clause to a try-catch block. The code inside the finally clause will always be executed, even if an exception is thrown from within the try or catch block. If your code has a return statement inside the try or catch block, the code inside the finally-block will get executed before returning from the method.

No matter whether an exception is thrown or not inside the try or catch block the code inside the finally-block is executed.

2. Stage **a1** (apply)

Lab Activities

Activity 1:

The example below shows the procedure for catching `IndexOutOfBoundsException` and `InputMismatch` exception.

. Solution:

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class Marks {
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        try{
            int[] marks = new int[5];
            for(int i=0;i<=5;i++){
                System.out.println("Enter marks for "+i+"st subjects :");
                marks[i]=s.nextInt();
            }
        }

        catch(InputMismatchException h){
            System.out.println("Please enter correct
            number!");
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("The error is"+e);
        }
    }
}
```

Activity 2:

The example below shows the procedure for using `throw` keyword.

Solution:

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

            throw new Exception();
        }
    }
}
```

```

        catch(Exception e){
            System.out.println("Error!!!");
        }
    }
}

```

Activity 3:

In some cases while developing our own applications, we need to specify custom types of exceptions to handle custom errors that may occur during program execution. A custom exception is a class that *inherits* Exception class or any of its subclasses, since inheritance is beyond the scope of this course, we will define it as using extends Exception in class declaration.

Solution:

```

class MyCustomException extends Exception{

    private String message;
    public MyCustomException(String message){
        this.message = message;
    }
    public String toString(){
        return message;
    }
}

```

Activity 4:

The example below shows the procedure for using throws keyword.

Solution:

//The following class represents the exception to be thrown:

```

public class illegalamount extends Exception {

    public illegalamount(){
        super();
    }

    public illegalamount(String a){
        super(a);
    }
}

```

```
}
```

//Account Class' methods deposit and withdraw will throw an exception of type illegalamount if the amount variable is not in the valid range.

```
public class Account {  
  
    private double balance;  
  
    public Account()  
    {  
        balance = 0;  
    }  
    public Account( double initialDeposit)  
    {  
        balance = initialDeposit;  
    }  
    public double getBalance()  
    {  
        return balance;  
    }  
  
    public double deposit( double amount)throws illegalamount{  
        if (amount > 0){  
            balance += amount;}  
  
        else{  
            throw new illegalamount("error in deposit");  
        }  
        return balance;  
    }  
  
    public double withdraw(double amount) throws  
    illegalamount {  
        if ((amount > balance) || (amount < 0)){  
            balance -= amount;}  
        else{  
            throw new illegalamount("error in withdraw");  
        }  
        return balance;  
    }  
}
```

3. Stage v (verify)

Home Activities

Activity 1:

Create a new Java file and:

Create an exception class named `NegativeNumberException` extended from the class `Exception`. This class should contain a parameterless constructor to call the `Exception` **class constructor with the message “You should enter a positive number”**.

Create a class named `exercise3` that contains:

A method named `M1` that accepts a double variable as argument and returns its square root. The method should throw the `NegativeNumberException` exception defined above.

The main method to prompt the user to enter a number and then to call the method `M1` (insert a try-catch block to handle the `NegativeNumberException` exception)

Activity 2:

Write java code, which throws arithmetic exception, with one catch block.

Activity 3:

Write java code, which throws null Pointer exception, with two catch block.

Activity 4:

Write java code, which throws index out of bound exception, with three catch block and one final block.

Activity 5:

Code of `Triangle` class is given below. **`Triangle`** class is defined with three sides. In a triangle, the sum of any two sides is greater than the other side. The **`Triangle`** class must adhere to this rule. Create the **`IllegalTriangleException`** class, and modify the constructor of **`Triangle`** class to throw an **`IllegalTriangleException`** object if a triangle is created with sides that violate the rule, as follows:

```
//Constructor
```

```
public Triangle(double side 1, double side 2, double side 3) throws  
IllegalTriangleException {
```

```
//Implement it
```



```
}
```

The constructor should not handle the exception itself. Rather, it should propagate the exception to the method in which the constructor is called (i.e. the triangle object is created).

```
public class Triangle {
    private double side1, side2, side3;

    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    public double getSide1() {
        return side1;
    }

    public void setSide1(double side1) {
        this.side1 = side1;
    }

    public double getSide2() {
        return side2;
    }

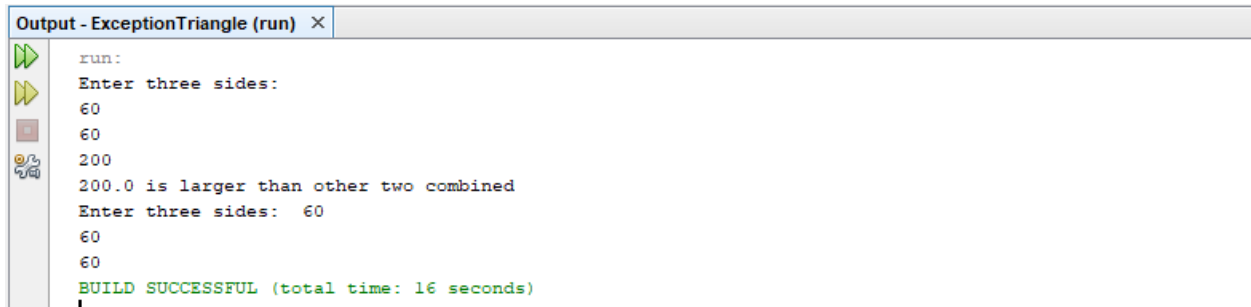
    public void setSide2(double side2) {
        this.side2 = side2;
    }

    public double getSide3() {
        return side3;
    }

    public void setSide3(double side3) {
        this.side3 = side3;
    }

    @Override
    public String toString() {
        return "Triangle [getSide1()=" + getSide1() + ", getSide2()=" + getSide2() + ", getSide3()=" +
            getSide3() + "]";
    }
}
```

Expected Output



```
run:
Enter three sides:
60
60
200
200.0 is larger than other two combined
Enter three sides: 60
60
60
BUILD SUCCESSFUL (total time: 16 seconds)
```

4. Stage a2 (access)

Assignment

Task 1:

Create a UserService Interface with following methods.

void addUser(User user)

void updateUser(User user)

void deleteUser(int userID)

User getUser(int userID)

void unblockUser(int userID)

void blockUser(int userID)

ArrayList<User> getAllUsers()

Define InMemoryUserService class that shall implement UserService contract or interface. Use ArrayList for storage, update and retrieve operations inside InMemoryUserService i.e. you shall define a static attribute ArrayList<User> users = new ArrayList<>(); The implemented methods shall change/read this list to perform the required operation. Test all methods from UserTest class. Getting information from user input is encouraged but optional.

Define at least id, name, and status instance attributes in User class of type int, String

and Boolean. How you check whether a user account is blocked? If status attribute is false, it means user account is blocked. True represent the account is active.

Make a test class to test the functionality of UserService ... no need to take inputs from user. Just call its different methods to see how it works, keep printing related information on console to show what the program is doing.

Improve above code using exception handling as explained below.

Change addUser and getUser methods declaration in the interface as follows:

void addUser(User user) throws UserAlreadyExistException

User getUser(int userID) throws UserAccountIsBlockedException

UserAlreadyExistException exception shall be thrown if user already exist in the *users* array list. The exception object shall also store the user ID for which the exception occurred. Check using *equals* methods (override the method in User class, based on user ID). Make *UserAlreadyExistException* a runtime exception. *UserAccountIsBlockedException* should be thrown if status attribute of the user object (in the array list) is false. Define this exception as checked.

Update the test class by handling both exceptions. Call different methods passing parameters that shall make the user service to throw the exception to demonstrate you handled them appropriately.

Statement Purpose:

The purpose of lab is to make students understand ways of getting information in and out of your Java programs, using files.

Activity Outcomes:

Students will be able to retrieve create file.

Students will be able to write and read data to and from a file.

Students will learn about Object Serialization.

Instructor Note:

The student should be able to declare a class.

The student should have understanding about Exception Handling.

1) Stage I (Journey)

Introduction

To read an entire object from or write an entire object to a file, Java provides object

serialization. A serialized object is represented as a sequence of bytes that includes the **object's data and its type information**. **After a serialized object has been written** into a file, it

can be read from the file and deserialized to recreate the object in memory.

A class that implements the Serializable interface is said to be a **serializable** class. To use objects of a class with writeObject() and readObject() , that class must be serializable. But to make the class serializable, we change nothing in the class. All we do is add the phrase implements Serializable . This phrase tells the run-time system that it is OK to treat objects of the class in a particular way when doing file I/O.

Classes ObjectInputStream and ObjectOutputStream, which respectively implement the ObjectInput and ObjectOutput interfaces, enable entire objects to be read from or written to a stream.

To use serialization with files, initialize ObjectInputStream and ObjectOutputStream objects with FileInputStream and FileOutputStream objects

ObjectOutput interface method writeObject takes an Object as an argument and writes its information to an OutputStream.

A class that implements ObjectOutput (such as ObjectOutputStream) declares this method and ensures that the object being output implements Serializable.

ObjectInput interface method readObject reads and returns a reference to an Object from an

InputStream. After an object has been read, its reference can be cast to the object's actual

type.

```
. /**
    Demonstrates binary file I/O of serializable class objects.
 */

public class ObjectIODemo

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

            ObjectOutputStream outputStream =
                new ObjectOutputStream(new FileOutputStream("datafile"));
            SomeClassoneObject = new SomeClass(1, 'A');
            SomeClassanotherObject = new SomeClass(42, 'Z');
            outputStream.writeObject(oneObject);

            outputStream.writeObject(anotherObject);
            outputStream.close();

            System.out.println("Data sent to file.");
        }
        catch(IOException e)
```

```

    {
System.out.println("Problem with file output.");
    }

System.out.println(
    "Now let's reopen the file and display the data.");

try

    {
ObjectInputStream inputStream =

    new ObjectInputStream(new
FileInputStream("datafile")); Notice the type
casts.

SomeClass readOne = (SomeClass)inputStream.readObject( );

SomeClass readTwo = (SomeClass)inputStream.readObject( );
System.out.println("The following were read from the file:");
System.out.println(readOne);

System.out.println(readTwo);
    }
catch(FileNotFoundException e)
{
System.out.println("Cannot find datafile.");

}
catch(ClassNotFoundException e)
{
System.out.println("Problems with file input.");
}

```

```

catch(IOException e)

    {

System.out.println("Problems with file input.");

    }

System.out.println("End of program.");

    }

}

```

2) Stage **a1** (apply)

Lab Activities:

Activity 1:

The following example demonstrates writing of objects to file.

Solution:

```

import java.io.*;

public static class Person implements Serializable
{
    Public String name=null;

    Public int age=0

}

public class ObjectOutputStreamExample {

```



```

public void writeObject() {

    try
    {
        ObjectOutputStream objectOutputStream =

            new ObjectOutputStream(new FileOutputStream("filename"));

        Person p = new Person();
        person.name = "Jakob Jenkov";
        person.age = 40;

        objectOutputStream.writeObject(p);

    }
    catch (ClassNotFoundException ex) {
        ex.printStackTrace();

    } catch
    (FileNotFoundException
    ex) { ex.printStackTrace();

    } catch
    (IOException
    ex) {
        ex.printStackTr
        ace();

    }

    finally
    {

        //Close the OutputStream

        try {

```

```

        if (objectOutputStream!= null) {
            objectOutputStream.close();
        }
    } catch (IOException ex) {

        ex.printStackTrace();
    }
}

}

}

```

Activity 2:

The following example demonstrates reading of all objects from a file.

Solution:

```

import java.io.*;

public class ObjectInputStreamExample {

    public void readObjects()
    {

        try
        {

```

```

ObjectInputStream objectInputStream =
    new ObjectInputStream(new FileInputStream("filename"));

while (true)
{

    Person personRead = (Person)
    objectInputStream.readObject();
    System.out.println(personRead.name);
    System.out.println(personRead.age);

}

}

catch (EOFException ex) { //This exception will be caught when EOF is
    reached System.out.println("End of file reached.");

    } catch
    (ClassNotFoundException
    ex) { ex.printStackTrace();

    } catch
    (FileNotFoundException
    ex) { ex.printStackTrace();

    } catch
    (IOException
    ex) {
    ex.printStackTrace();
    }

    }

finally
{

    //Close the ObjectInputStream

```

```

        try {

            if (objectInputStream!= null) {

                objectInputStream.close();

            }

        } catch
        (IOException
         ex) {
            ex.printStackTrace();

        }

    }

}

}

}

}

```

Activity 3:

The following example demonstrates searching of an object from a file.

Solution:

```

import java.io.*;

public class ObjectInputStreamExample {

    public void searchObject(String name)
    {

        try

```

```

{

    ObjectInputStream objectInputStream =
        new ObjectInputStream(new FileInputStream("filename"));

    while (true)
    {

        Person personRead = (Person) objectInputStream.readObject();

        If (personRead.name.equals(name)
            {
                System.out.println(personRead.name);
                System.out.println(personRead.age);
                break;
            }

        }

    }

    catch (EOFException ex) { //This exception will be caught when EOF is
        reached System.out.println("End of file reached.");

        } catch (ClassNotFoundException
            ex) { ex.printStackTrace();

        } catch (FileNotFoundException
            ex) { ex.printStackTrace();

        } catch (IOException
            ex) {
                ex.printStackTrace();
            }

```

```

    }
finally
{
    //Close the ObjectInputStream
    try {
        if (objectInputStream!= null) {
            objectInputStream.close();
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
}
}
}

```

3) Stage **v** (verify)

Home Activities:

Activity 1:

Create a class Book that has name(String), publisher (String) and an author (Person).
Write **five objects of Book Class in a file named “BookStore”**.

Activity 2:

Consider the Book class of Activity 1 and write a function that displays all Books present in **file “BookStore”**.

Activity 3:

Consider the Book class of Activity 1 and write a function that asks the user for the name of a Book and searches the record against **that book in the file “BookStore”**.

4) Stage **a2** (assess)

Assignment:

Create an ATM System with **Account** as the Serializable class. Write ten objects of **Account** in a file. Now write functions for withdraw, deposit, transfer and balance inquiry.

Note:

- a. Each function should ask for the account number on which specific operation should be made.
- b. All changes in Account object should be effectively represented in the file.