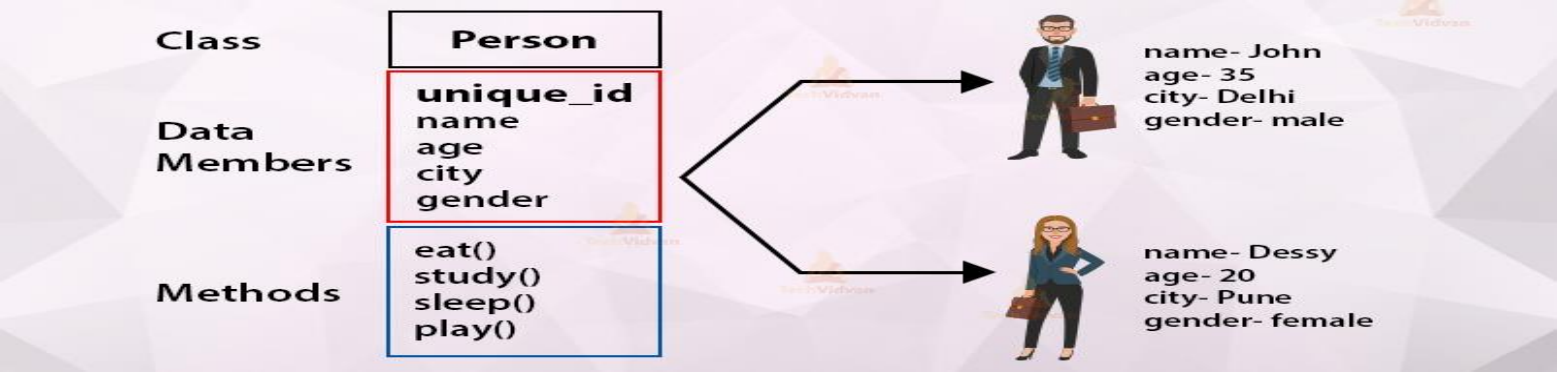
OBJECT ORIENTED PROGRAMMING

JAVA

Object Oriented Programming Concepts

- Aim of object-oriented programming is to **implement real-world entities.**
- A paradigm to design a program using **classes and objects**.
- **Object** means a real-world entity, for example, chair, table, pen, computer, watch, etc.
- **Class** means ***Collection of objects***. It is a logical entity.
- A blueprint from which you can create an individual object. It doesn't consume any space.



What is Java?

- Java is a general-purpose **object oriented programming language** and a platform.

- Java is a high level, robust, secured and object-oriented programming language.

- Works on **WORA(Write Once Run Anywhere) model**.

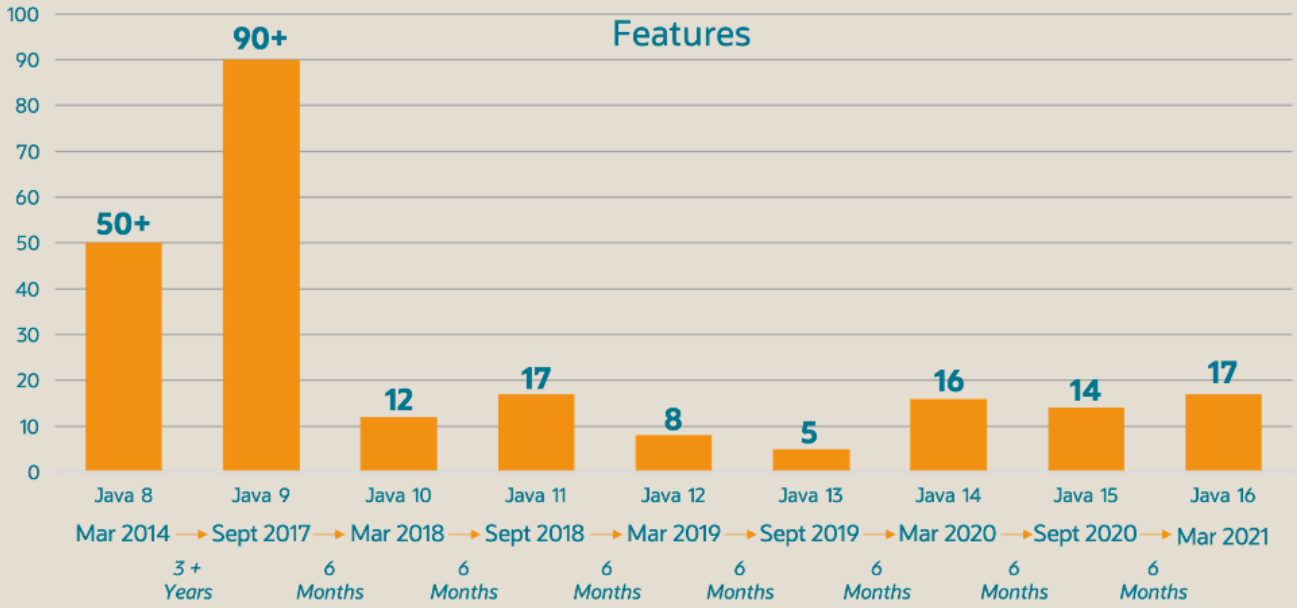
What is a platform?

- Any hardware or software environment in which a program runs, is known as a **platform**.

- Since Java has is own **runtime environment** (JRE) and API, it is called platform.

History of Java

- **James Gosling** - Sun Microsystems
- Oak - Java, May 20, 1995, Sun World
- JDK Evolutions



Significance of Java

- Two reasons :
  - Trouble with C/C++ language is that they are not portable and are not platform independent languages.
  - Emergence of World Wide Web, which demanded portable programs

- Portability and security necessitated the invention of Java

Where it is used?

- According to Sun, 3 billion devices run java.
- There are many devices where java is currently used. Some of them are as follows:
    - Desktop Applications
    - Web Applications
    - Enterprise Applications
    - Mobiles
    - Embedded Systems
    - Smart Card
    - Robotics
    - Games

Java Editions

- **Java platform** is a collection of programs that help to develop and run programs written in the Java programming language.

1) **Java SE (Java Standard Edition)**

2) **Java EE (Java Enterprise Edition)**

3) **Java ME (Java Micro Edition)**

4) **JavaFX**

Java Installation

- Please go to the link https://www.oracle.com/in/java/technologies/javase/jdk14-archive-downloads.html

- Download JDK as per OS, you are using (Linux/Mac/Windows)

- Then, follow the JDK Installation Instructions for Windows/Mac/Linux from below link

https://docs.oracle.com/javase/9/install/installation-jdk-and-jre-microsoft-windows-platforms.htm#JSJIG-GUID-DAF345BA-B3E7-4CF2-B87A-B6662D691840

- Once installation is done need to set path, which is mentioned in next slide.

Setting the path environment variable

There are two ways for setting the path:

1. Path can be set via the Command Prompt(To set Temporary Path)

- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=javabin_path

Command Prompt

```
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\S340>cd\

C:\>cd new

C:\new>set path="C:\Program Files\Java\jdk-14.0.2\bin"

C:\new>javac
Usage: javac <options> <source files>
where possible options include:
  @<filename>                  Read options and filenames from file
  -Akey[=value]                Options to pass to annotation processors
  --add-modules <module>(,<module>)*
        Root modules to resolve in addition to the initial modules, or all modules
        on the module path if <module> is ALL-MODULE-PATH.
  --boot-class-path <path>, -bootclasspath <path>
        Override location of bootstrap class files
  --class-path <path>, -classpath <path>, -cp <path>
        Specify where to find user class files and annotation processors
  -d <directory>               Specify where to place generated class files
  -deprecation
        Output source locations where deprecated APIs are used
```

Setting the path environment variable

Path can be set via the Command Prompt (Contd.)

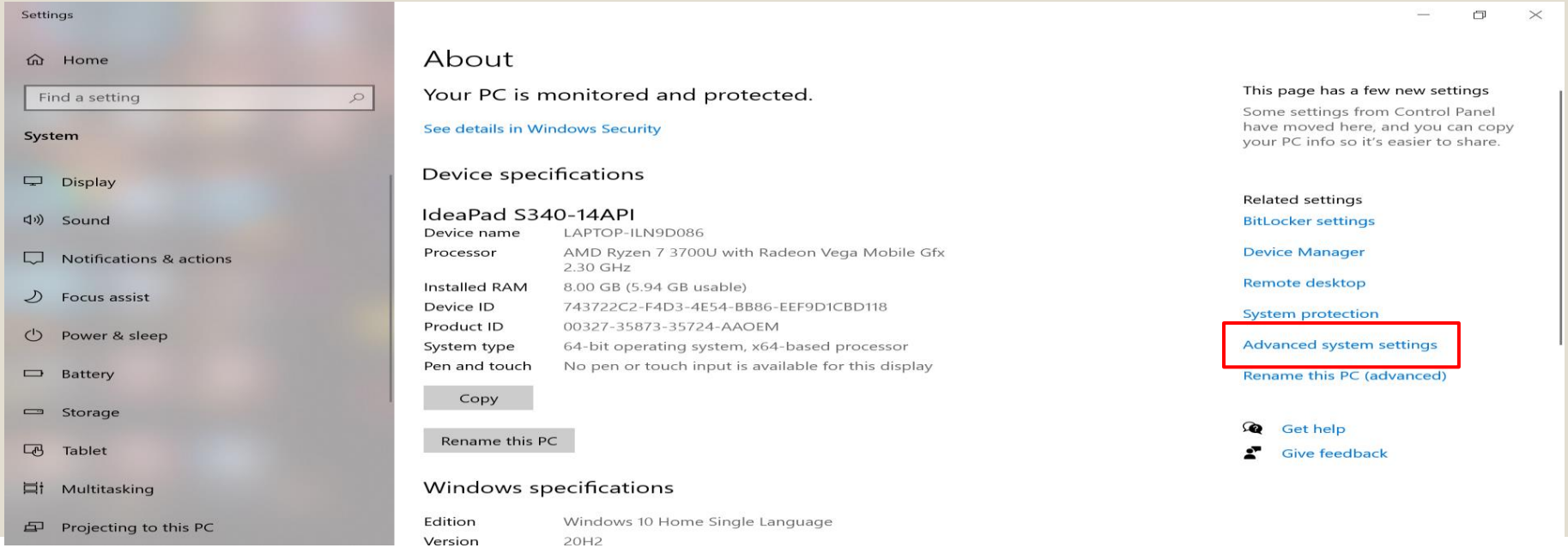
Command Prompt

```
C:\new>javac HelloWorld.java

C:\new>java HelloWorld
Hello, World

C:\new>_
```

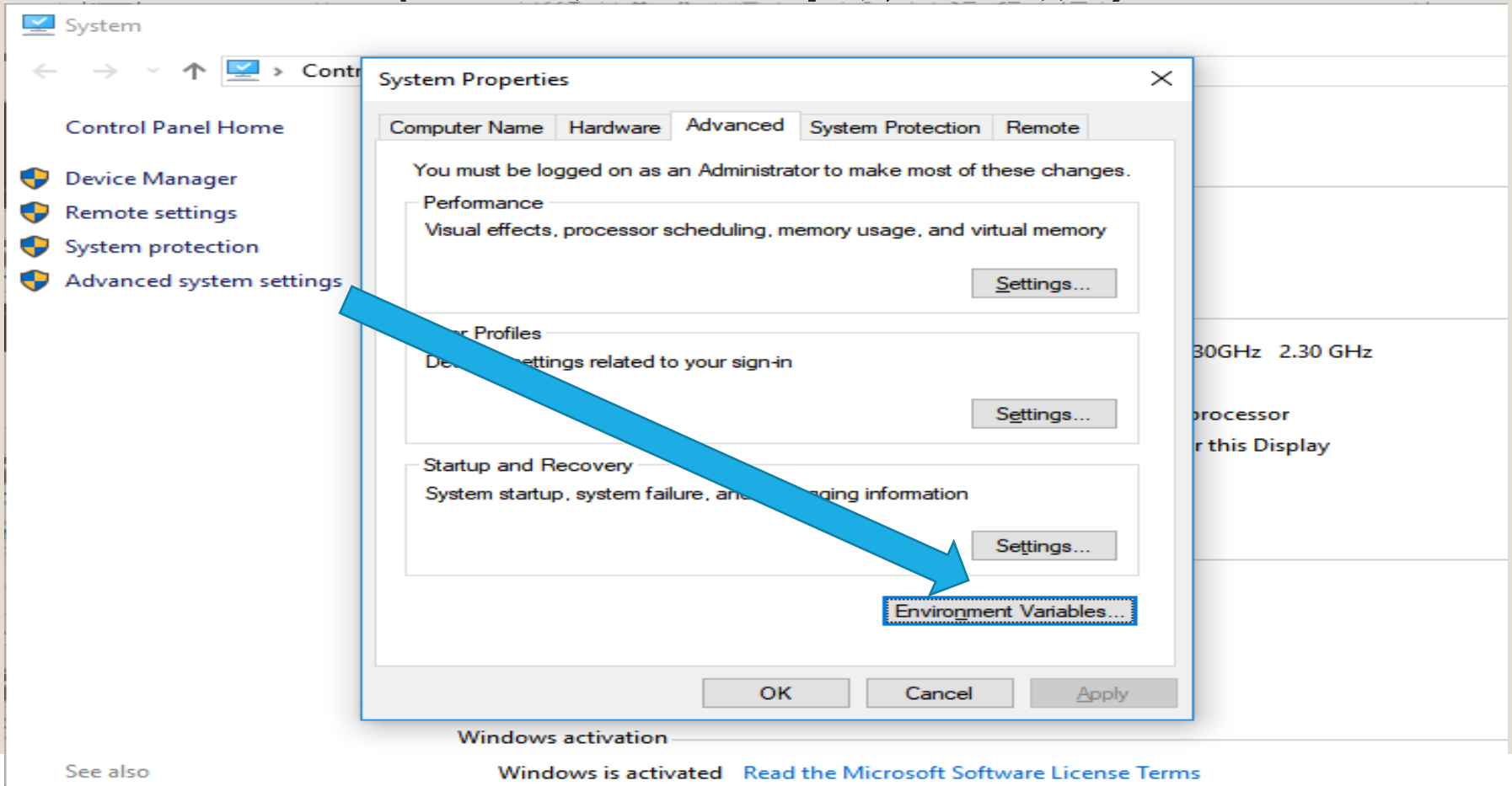
Setting the path environment variable

Path can be set via the Control Panel (To set Permanent Path)

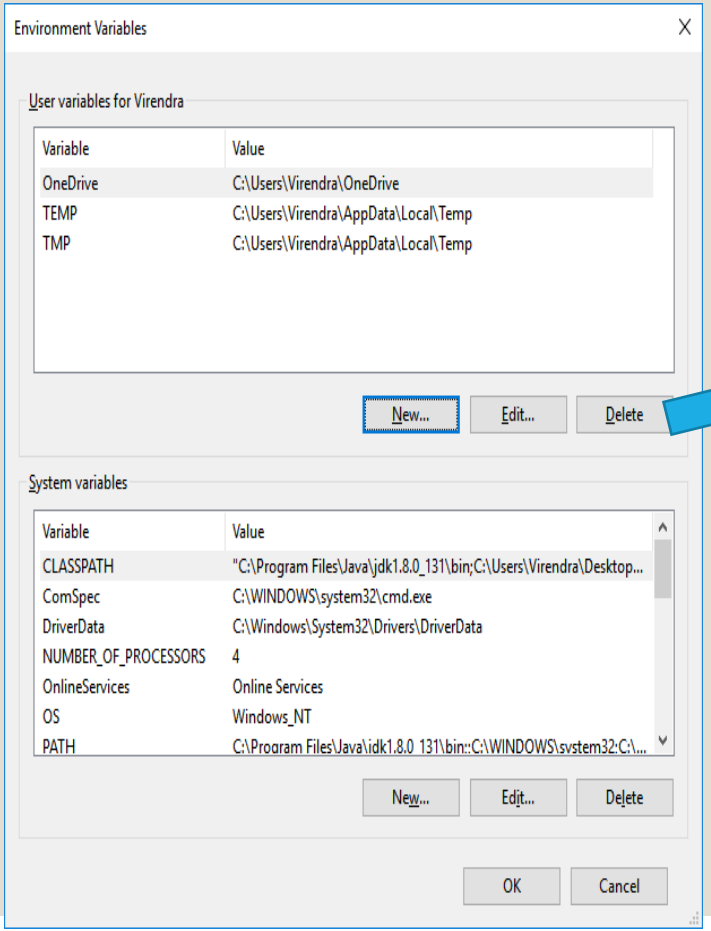
Go to My Computer properties -> Right hand side check ‘Advanced System Settings’ -> environment variables -> new tab of user variable -> write the path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

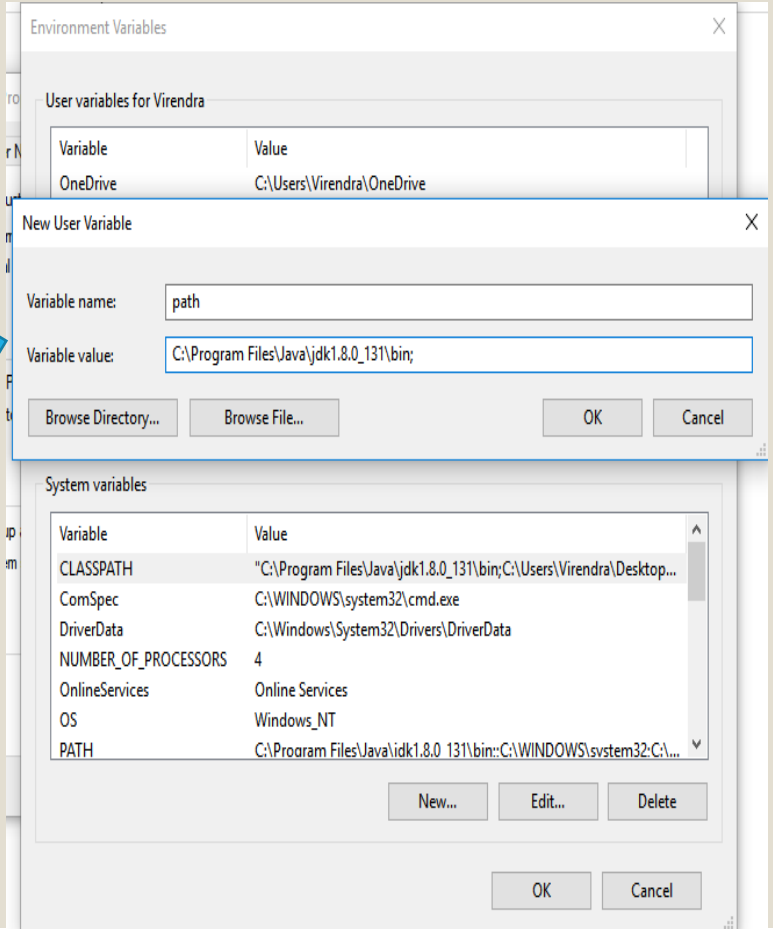


Setting the path environment variable (older version of Windows)

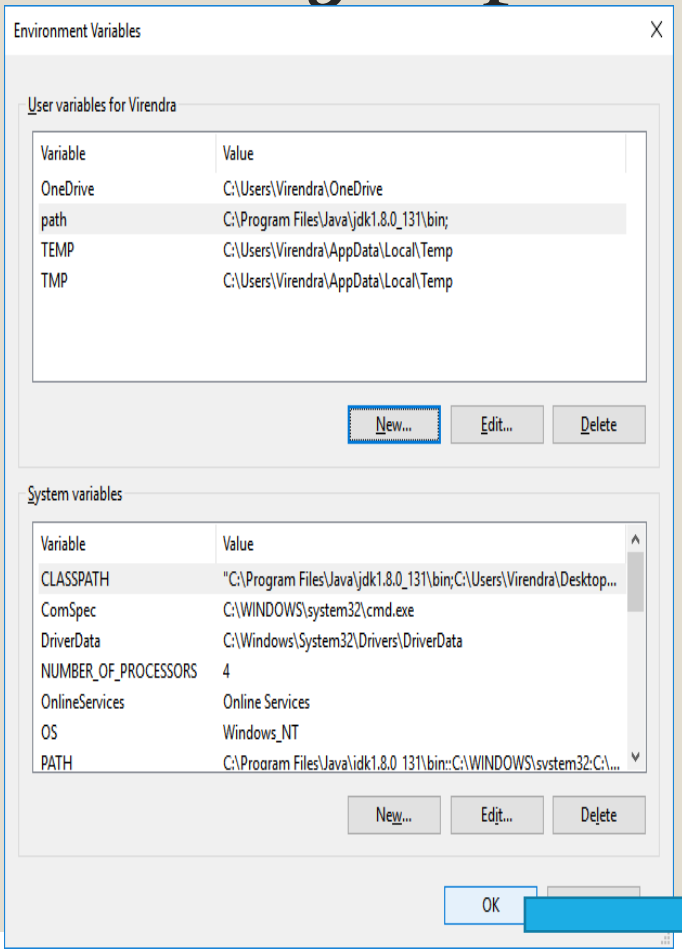


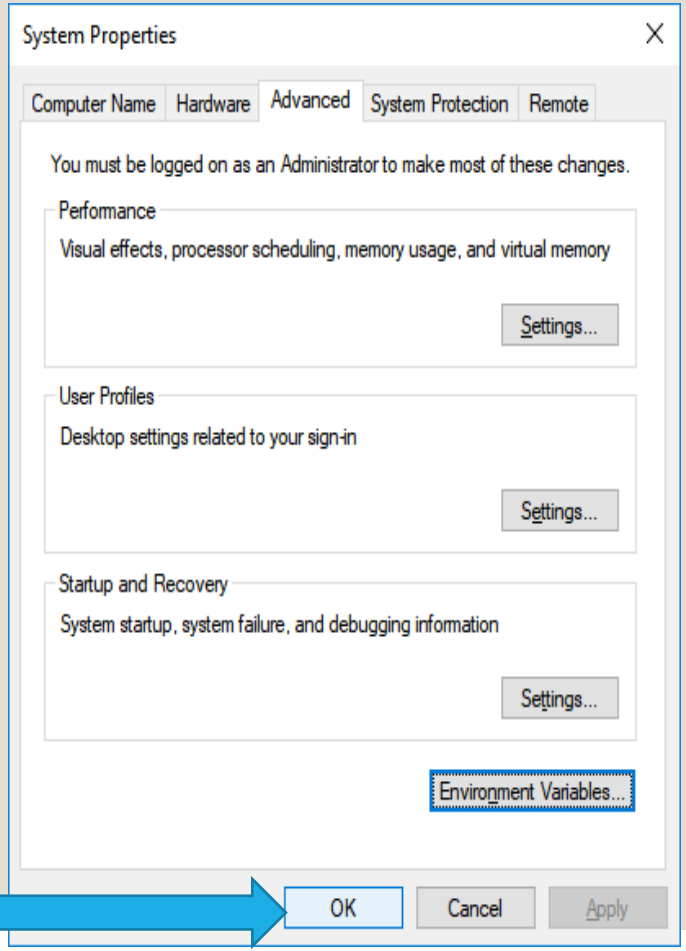
Setting the path environment variable





Setting the path environment variable





Java Program (HelloWorld.java)

```
import java.io.*;
public class HelloWorld
{
    public static void main(String[] args)
    {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }

}
```

classname

Main function

Comments

Print Statement

Setting the path environment variable

Command Prompt

```
C:\Users\Virendra>cd..

C:\Users>cd..

C:\>cd new

C:\new>javac HelloWorld.java

C:\new>java HelloWorld
Hello, World
```

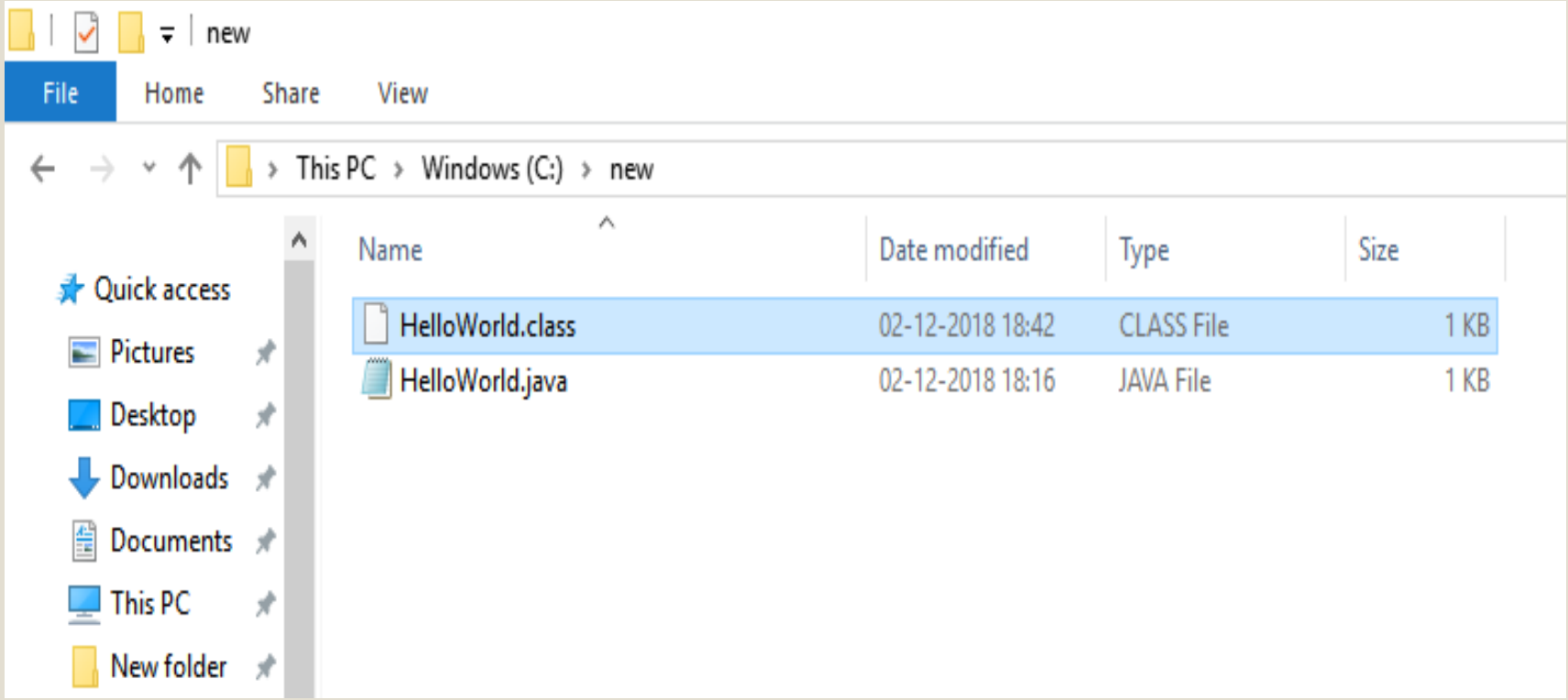
Output

Compiling and Interpreting Java Program

```
C:\Users\Virendra>cd\

C:\>cd new

C:\new>set path="C:\Program Files\Java\jdk1.8.0_131\bin"

C:\new>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation processors
  -cp <path>                 Specify where to find user class files and annotation processors
  -sourcepath <path>         Specify where to find input source files
```

```
C:\new>javac HelloWorld.java

C:\new>java HelloWorld
Hello, World
```

Compiling and Interpreting Java Program



Assignment: Practical No. 01

Aim: Ankit and his friends went out for a Pizza party. Ankit's friend asks him to cover the entire area of the Pizza with chilli flakes to have a strong hot taste. Compute the area and perimeter of Pizza using java program and print statement, where the diameter of Pizza is 20 cm.

Exercises

System.out.println(1+0+1+“ Hello”);

Output: 2 Hello

System.out.println(“Hello ”+1+0+1);

Output: Hello 101

System.out.println(“Hello ”+(1+0+1));

Output: Hello 2

Syntax & Examples

- datatype variable_name=value;
- **Examples:**

int num1=10, num2=1;

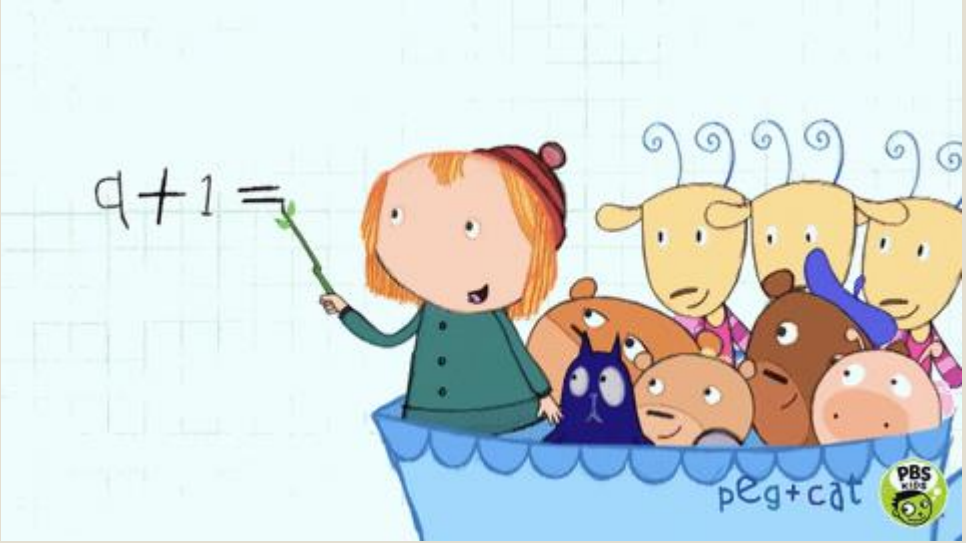
float z=6.2123;

double ax=2.33;

String name=“OOP”;

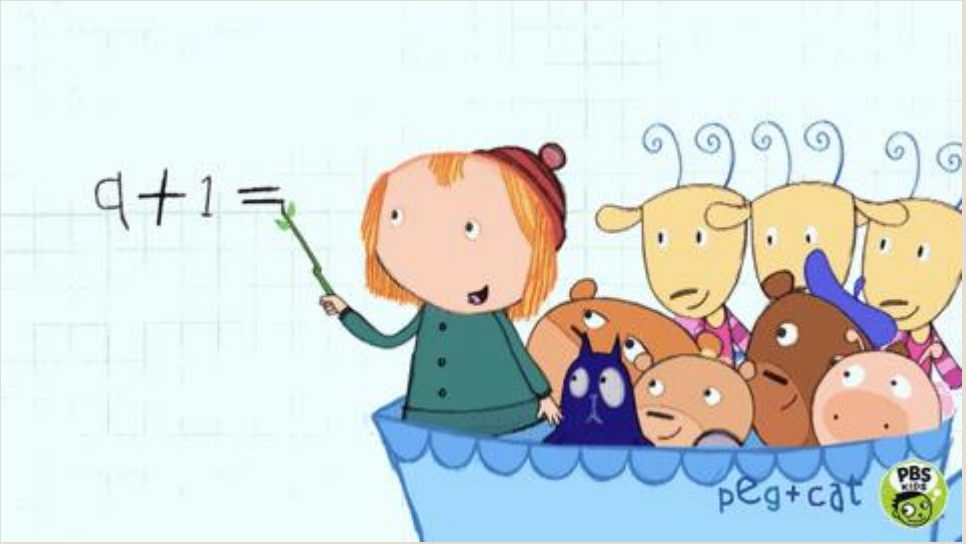
Practice

Aim: Aahana had her Mathematics subject final examination. Her teacher instructed the students to solve the equation using the formula: **3.0 * (1 + (4.0/4) + (2.0*5))**through a java program, and print the output.



Assignment: Practical No. 02

Aim: Aahana had her Mathematics subject final examination. Her teacher instructed the students to solve the equation using the formula: **2.0 * (1 - (3.0/3) + (1.0/5) - (2.0/14) + (1.0/12) - (2.0/22))** through a java program, and print the output.



return statement

- To return a value from function, we use the keyword **return.**
- We can return a direct console value of variable value. But, a function can return only one value.
- **return statement should be the last statement of a function, because it also returns the controls back to the calling function.**
- The datatype of returned value will be the return-type of that function.
- If a function does not return anything, then return-type will be void.
- **Syntax:**

return value/variable_name;

Example:

```
return z;
return 12;
```

Functions

Syntax to create a function:

```
return-type function_name
{
   //block of code
}
```

Example 1:

```
void getData()
{
  System.out.println("Hi");
}
```

Example 2:

```
int getData()
{
   int x=11;
   return x;
}
```

Example 3:

```
double passInfo()
{
     return 11.22;
}
```

Assignment: Practical No. 03

Aim: Aditya went to a supermarket to purchase 12 pens worth Rs. 144 and a set of 6 books costing Rs. 120. Compute the total cost incurred by Aditya in purchasing the stationary items, that is, the total invoice to be generated by the shopkeeper through java program implementation.

NetBeans Installation

Practice

Aim: Ojaswitaa's company assigned her to develop a module in a number validation system. The client instructed her to design the module which will verify whether a number entered is positive or negative using a Java program implementation.

Practice Solution

```
public class PosNegNum
{
    public static void main(String[] args)
    {
      int x=10;
      if(x>0)
      {
          System.out.println(x+" is positive");
      }
      else if(x<0)
      {
          System.out.println(x+" is negative");
      }
else
      {
          System.out.println(x+" is zero");
      }
    }

}
```

Practice Alternate Solution using Objects

```
public class PosNegNum
{
    void verify()
    {
      int x=10;
      if(x>0)
      {
          System.out.println(x+" is positive");
      }
      else if(x<0)
      {
          System.out.println(x+" is negative");
      }
      else
      {
          System.out.println(x+" is zero");
      }
```

```
    public static void main(String[] args)
    {
        PosNegNum p=new PosNegNum();
        p.verify();
    }

}
```

User Input on a Console

- One really useful class that handles input from a user is called the **Scanner** class. The Scanner class can be found in the **java.util** library. To use the Scanner class, you need to reference it in your code:

import **java.util.Scanner;**

- The next thing you need to do is to **create an object from the Scanner class.** To create a new Scanner object the code is this:

Scanner a = new Scanner(System.in);

Here Scanner is the class name, a is the name of the object, new keyword is used to allocate the memory and System.in is the input stream.

User Input on a Console

- To **get the user input**, you can call into action one of the many methods available to your new Scanner object. One of these methods is called next(). This gets the next string of text that a user types on the keyboard:

```
String stream_name;

Scanner course_input=new Scanner(System.in);
stream_name = course_input.next( );
```

- Following methods of Scanner class are used in the program:

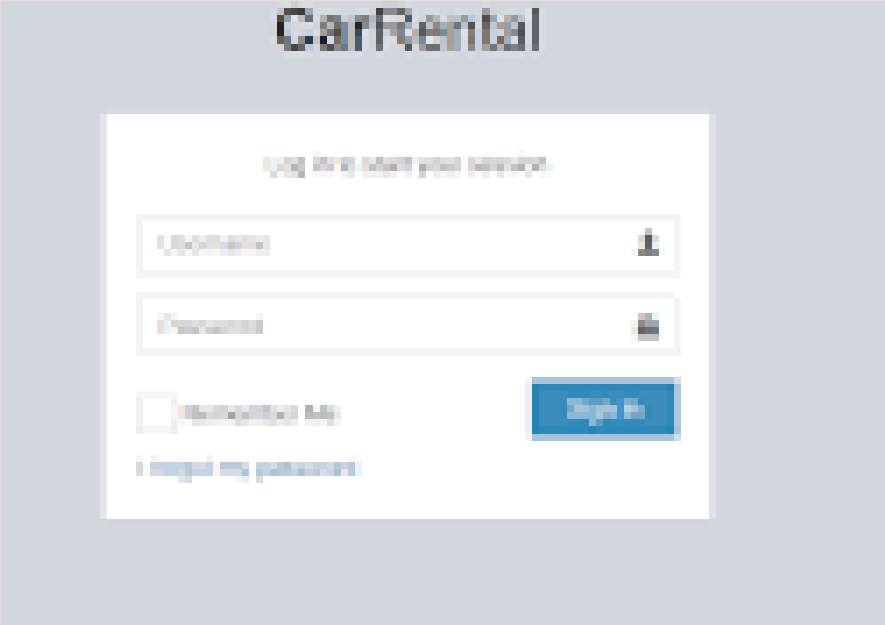
- **next() or nextLine() to input a string**
- **nextInt() to input an integer**
- **nextFloat() to input a float**

Practice Alternate Solution using Objects

```
import java.util.*;
public class PosNegNum
{
    void verify()
    {
      Scanner sc=new Scanner(System.in);
      int x=sc.nextInt();
      if(x>0)
      {
          System.out.println(x+" is positive");
      }
      else if(x<0)
      {
          System.out.println(x+" is negative");
      }
      else
      {
          System.out.println(x+" is zero");
      }
    }
    public static void main(String[] args)
    {
      PosNegNum p=new PosNegNum();
      p.verify();
    }

}
```

Assignment: Practical No. 04

Aim: Rajan is designing a car rental website which requires the customer to log in to the website for renting a car using a java program to input the login details such as username, password and print the message as ‘Welcome Username’ or ‘Failed to login!! Please try again’ depending on successful or failure in login.



Assignment: Practical No. 05

Aim: A company received a project to develop an employee management system. Neha was assigned to develop a module of taking the user input of employee details such as name, age and salary. Design a program to accept input type in string, integer and double datatype respectively and display the details of the employee.

Assignment: Practical No. 06

Aim: A bank management system is being developed which requires taking input of the customer id, customer name of a specific bank. Consider the bank name and account balance as a static variable through Java implementation and display the values.

Practice

Program 1:

Aim: A college university wants to generate a marksheet designing system using java where the enrollment id, name of the student, three subject marks should be taken as input and displayed along with the average of the three subjects.

Program 2:

Aim: A mathematical application is to be developed for a student training program with inbuilt calculator system. Design a calculator application using java using classes and objects.

Practice: Polymorphism

Aim: Riya created a calculator to support addition of 2 numbers, 3 numbers and 4 numbers using java program through method overloading.

Assignment: Practical No. 07

Aim: Ravi and his friends are playing a multiplayer game which requires each player to compute the area of a shape that is displayed in real time. Thus, overload the function compute() to print the area of different shapes (square, rectangle, circle) using the concept of polymorphism.

Array

- An array is a **collection of similar type** of elements that have a contiguous memory location.
- Java **array is an object** which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in java **is index-based**, the first element of the array is stored at the 0 index.
- **Syntax:**

```
datatype array_name[]={value1,value2,….};                    //Array
```

Example:

```
int a[]={111,32,411,15};
```

Practice: Basic Array

Aim: Aashvika randomly inputs following values for sorting in two different variables: **11,33,22,44,88,77,99,66** and **Java, Python, PHP, C#, C Programming, C++**. Design a basic application to sort and display those values.

```
run:
Original numeric array : [11, 33, 22, 44, 88, 77, 99, 66]
Sorted numeric array : [11, 22, 33, 44, 66, 77, 88, 99]
Original string array : [Java, Python, PHP, C#, C Programming, C++]
Sorted string array : [C Programming, C#, C++, Java, PHP, Python]
```

Practice: Solution

```
package arraydata;
import java.util.*;
public class ArrayData
{
    public static void main(String[] args)
    {
      int[] num_array1 = {11,33,22,44,88,77,99,66};
    String[] var_array2 = {"Java", "Python", "PHP", "C#", "C Programming", "C++"};
    System.out.println("Original numeric array : "+Arrays.toString(num_array1));
    Arrays.sort(num_array1);
    System.out.println("Sorted numeric array : "+Arrays.toString(num_array1));
    System.out.println("Original string array : "+Arrays.toString(var_array2));
    Arrays.sort(var_array2);
    System.out.println("Sorted string array : "+Arrays.toString(var_array2));
    }
}
```

Assignment: Practical No. 08

Aim: Aashka is designing a module for an office management system which will help her colleagues reduce their work by detecting the duplicate values in their data. Design a java program to implement the concept of array and print duplicate values from two arrays, if any.

```
run:
Array1 : [1, 2, 5, 5, 8, 9, 7, 10]
Array2 : [1, 0, 6, 15, 6, 4, 7, 0]
Common elements is/are : 1  7  BUI
```

Practice: Solution

```
package arraydata;
import java.util.Arrays;
public class ArrayDuplicate
{
public static void main(String[] args)
{
    int[] array1 = {1, 2, 5, 5, 8, 9, 7, 10};
    int[] array2 = {1, 0, 6, 15, 6, 4, 7, 0};

     System.out.println("Array1 : "+Arrays.toString(array1));
     System.out.println("Array2 : "+Arrays.toString(array2));
     System.out.print("Common elements is/are : ");
```

```
for (int i = 0; i < array1.length; i++)
    {
        for (int j = 0; j < array2.length; j++)
        {
            if(array1[i] == (array2[j]))
            {
             System.out.print((array1[i])+"  ");
             }
        }
    }

  }
}
```

Practice: ArrayCopy

```
package arraydata;


import java.util.*;


public class Arraycopy
{
  int[] old_array = {25, 14, 56, 15, 36, 56, 77, 18, 29, 49};
  int[] new_array;
  void getD()
  {
     Arrays.sort(new_array);
      System.out.println("SourceArray :
"+Arrays.toString(old_array));
      new_array=new int[10];
 for(int i=0; i < old_array.length; i++)
       {
              new_array[i] = old_array[i];
       }
       Arrays.sort(new_array);
       System.out.println("NewArray:
"+Arrays.toString(new_array));
  }
 public static void main(String[] args)
 {

  Arraycopy c;
  c=new Arraycopy();
  c.getD();

}

}
```

Assignment: Practice

Aim: Arjun was assigned a task to search for certain values in a worksheet. Owing to multiple values, it was difficult for him to search a data, develop a java program to help him implement the logic in java to search for an element in an array.

```
run:
Array contains 1456: true
Array contains 2444: false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Assignment: Practice

```
package arraydata;

public class ArrayFindElement
{
  public static boolean contains(int[] arr, int item) {
      for (int n : arr)
      {
         if (item == n)
         {
            return true;
         }
      }
      return false;
   }
   public static void main(String[] args)
   {
         int[] arr = {1789, 2035, 1899, 1456, 2013, 1458, 2458, 1254, 1472, 2365, 1456, 2265, 1457, 2456};
         System.out.println("Array contains 1456: "+(contains(arr, 1456)));
         System.out.println("Array contains 2444: "+(contains(arr, 2444)));
   }
}
```

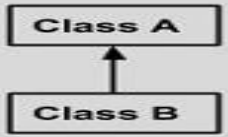
Assignment: Practical No. 09

Aim: Rehana has her mathematics assignment deadline today, where she is asked to compute the addition of following values [12,24,1,4] and [2,12,13,21]. Design a java program to help her complete the assignment through a java program implementation.

```
run:
Input number of rows of matrix
2
Input number of columns of matrix
2
Input elements of first matrix
12
24
1
4
Input the elements of second matrix2
12
13
21

Sum of the matrices:-
14        36
14        25
```

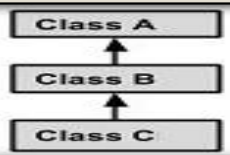
Types of Inheritance

Single Inheritance



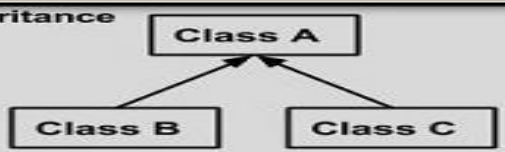

```
public class A {
  -------
}
public class B extends A {
  ---------
}
```

Multi Level Inheritance



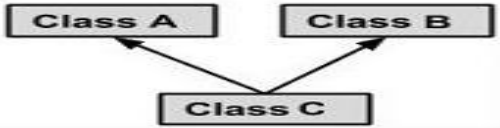

```
public class A { ------------------}
public class B extends A {------------------}
public class C extends  B {------------------ }
```

Hierarchical Inheritance




```
public class A { ------------------}
public class B extends A {------------------}
public class C extends A {------------------ }
```

Multiple Inheritance




```
public class A { ------------------}
public class B {------------------}
public class C extends  A,B {
  -------------------
} // Java does not support mutiple Inheritance
```

Practical No. 10

Aim: Zahir wants to fill an online admission form of a college, but requires the details of college such as college name and it's id. Implement a java based program to fetch the details and display them through single inheritance.

```
run:
Enter College Name:Ganpat
****College Details****
College name: Ganpat
College id: 1001
BUILD SUCCESSFUL (total time: 8 seconds)
```

Practical No. 10

```java
package inheritance;
import java.util.*;

class College
{
    protected int college_id=1001;
    String college_name;
    protected void getData()
    {
        System.out.print("Enter College Name:");
        Scanner sc=new Scanner(System.in);
        college_name=sc.next();
    }
}
```

Practical No. 10(Contd.)

```
class SingleInherit extends College
{
    void displayData()
    {
        getData();
        System.out.println("****College Details****\nCollege name: "+college_name+"\nCollege id: "+college_id);
    }
  public static void main(String args[])
  {
      SingleInherit si=new SingleInherit();
      si.displayData();


  }
}
```

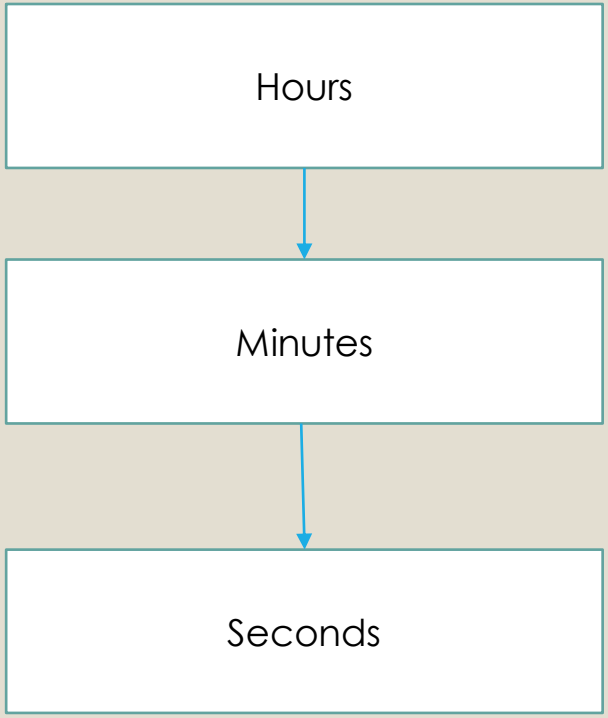
Practice

Create a java program to implement inheritance with the following details:

1. Base class: Stream (user input for stream)
2. Derived class: DBMS (user input for DBMS marks)
3. Derived class: OOP (user input for OOP marks)

Calculate the average marks of the two subjects and display their respective marks.

Practice



Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as **method overriding**.

❑ Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for **runtime polymorphism**.

❑ Rules for Java Method Overriding

- The method must have the same name as in the parent class.
- The method must have the same parameter as in the parent class.
- There must be an **IS-A relationship** (inheritance).

Practical No. 11

Aim: Karan wants to opt for a student loan to pursue further +2 years of education abroad. Implement a java program which surveys various banks for their rate of interest, and display the returned ROI's through hierarchical inheritance. Consider the ROI of banks are as follows:

SBI	**6.9%**
HDFC	**7.1%**
BOI	**6.5%**

```
run:
SBI Rate of Interest: 6.9
HDFC Rate of Interest: 7.1
BOI Rate of Interest: 6.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
package inheritance;

public class HierarchicalInherit
{
public static void main(String[] args)
{
      Bank b;

        b=new SBI();
        System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
        b=new HDFC();
        System.out.println("HDFC Rate of Interest: "+b.getRateOfInterest());
        b=new BOI();
        System.out.println("BOI Rate of Interest: "+b.getRateOfInterest);
    }
}
```

Practical No. 11

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{

float getRateOfInterest()
{return 6.9f;}
}
class HDFC extends Bank{

float getRateOfInterest(){return 7.1f;}
}
class BOI extends Bank{
float getRateOfInterest()
{
    return 6.5f;
}
}
```

Abstract Class

- A class which is declared with the **abstract keyword** is known as an **abstract class** in Java. It can have **abstract and non-abstract methods (method with the body).**
- It needs to be extended and its method implemented. It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.
- Example:

```
abstract class A{}
```

❑ Abstract Methods in Java

- A method which is **declared as abstract and does not have implementation** is known as an abstract method.

```
abstract void printStatus();          //no method body and abstract
```

Abstract Class(Contd.)

Example:

```
abstract class Bike
{
    abstract void move();
}
class Vehicle extends Bike
{
    void move()
    {
        System.out.println("Accelerated safely");
    }
    public static void main(String args[])
    {
        Bike obj = new Vehicle();
        obj.move();
    }
}
```

Practical No. 12

Aim: A vehicle showroom management system displays the speed of two wheelers and four wheelers as 90 km/hr and 180 km/hr respectively, implement the same using abstract class in java.

Abstract class: Vehicle

Abstract function: getSpeed()

Derived Class: TwoWheeler
Function: getSpeed() : 90km/hr

Derived Class: FourWheeler
Function: getSpeed() : 180km/hr

```
run:
Speed of two wheeler is 90 km/hr
Speed of four wheeler is 180 km/hr
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abstract Class(Contd.)

Example:

```
abstract class Bike
{
  Bike()
{System.out.println("bike is created");}
  abstract void run();
  void changeGear()
 {System.out.println("Gear changed");}
 }
 class Honda extends Bike
{
 void run()
{System.out.println("running safely..");}
 }
```

```
class TestAbstraction2
{
 public static void main(String args[])
{
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Abstract class: Bike
Constructor
Abstract function: run()
Non-abstract function: changeGear()

Derived Class: Honda
Function: run()

```
run:
Bike is created
Running safely..
Gear changed
```

Interface in Java

- An **interface in java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**
- It cannot be instantiated just like abstract class.

Why use Java interface?

- There are mainly three reasons to use interface. They are given below.

➢ It is used to **achieve abstraction.**

➢ By interface, we can support the **functionality of multiple inheritance.**

➢ It can be used to **achieve loose coupling.**

❑ **Declaring an interface**

➢ An interface is declared by using the **interface keyword**.

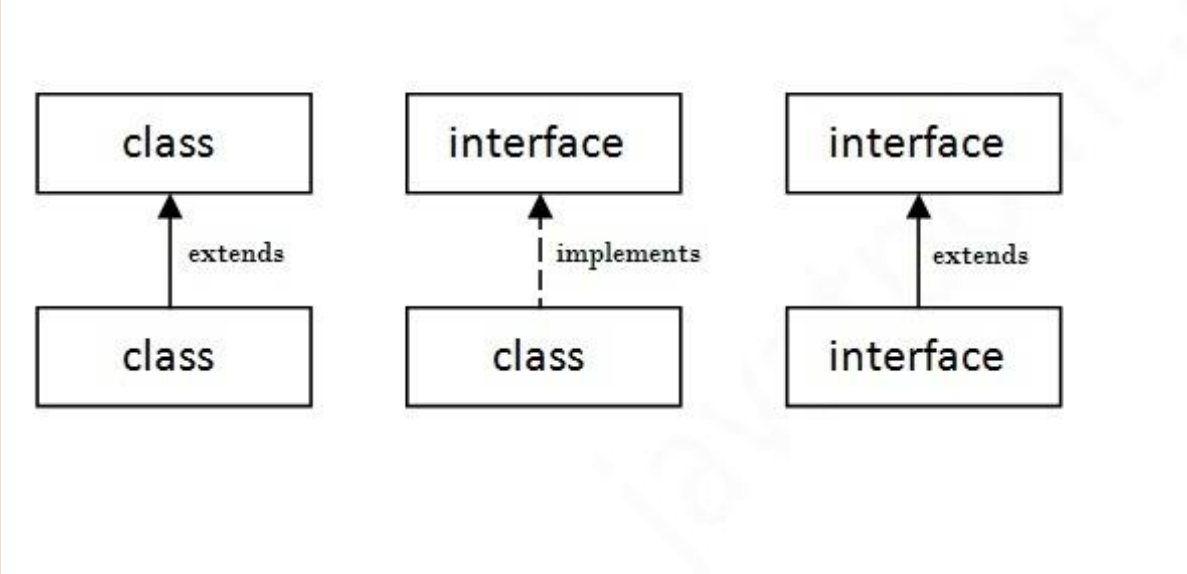
➢ It provides **total abstraction**; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.

➢ A class that implements an interface must implement all the methods declared in the interface.

Understanding relationship between classes and interfaces

Syntax:

```
interface <interface_name>
{
   // declare constant fields
   // declare methods that abstract
   // by default.
}
```



Example: Practice

```
package inheritance;
interface MyInterface
{
   public void method1();
   public void method2();
}
class Demo implements MyInterface
{
  @Override
   public void method1()
   {

            System.out.println("Method1 Implementation");
   }
   @Override
   public void method2()
   {

            System.out.println("Method2 Implementation");
   }
}
```

```
public class InterfaceDemo
    {
        public static void main(String[] args)
        {
            MyInterface obj = new Demo();
            obj.method1();
            obj.method2();
        }
    }
```

```
run:
Method1 Implementation
Method2 Implementation
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical No. 13

Aim: A zoo management system maintains details about different animals with their respective features. Using abstraction, display the features such as speaking, number of legs of animals.

Interface1: Animals
Abstract function: speak()

Interface2: Paws
Abstract function: noOfLegs()

Derived Class: Dog

```
run:
Dog Barks
They have 4 legs
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical No. 14

Aim: A library management software is to be designed to ease the fetching of books based on their title present within the library. Implement complete abstraction through interface and abstract class:

i. Abstract class Book to assign a title through parameters

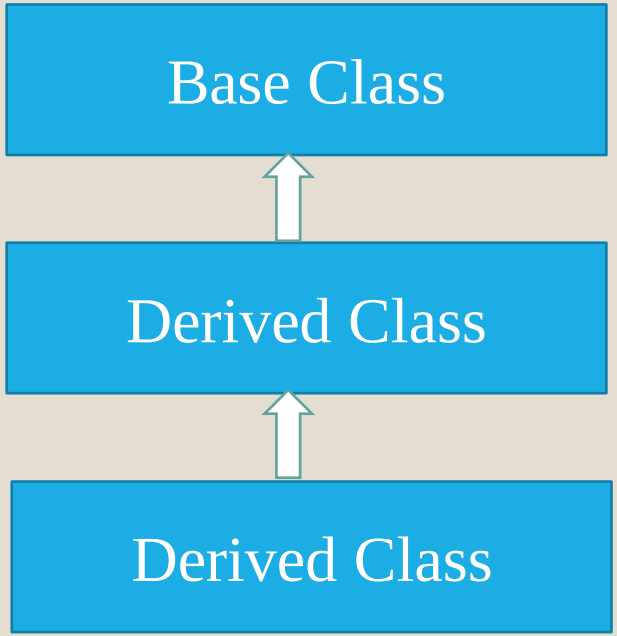
ii. Interface TitleBook to fetch a title

iii. MyBook to override the abstract functions

```
run:
Java Fundamentals
The title is: Java Fundamentals
BUILD SUCCESSFUL (total time: 9 seconds)
```

Question

Aim: Add if statement to InheritStore. Get user input for discount. If discount is less than 10% then print discount of it is more than 10% print accordingly.



PRACTICE

Aim: Complete the code mentioned by writing an overridden getNumberOfTeamMembers method that prints the same statement as the superclass' getNumberOfTeamMembers method, except that it replaces n with 11 (the number of players on a Soccer team).

Output Format:

When executed, your completed code should print the following:

Generic Sports
Each team has n players in Generic Sports
Soccer Class
Each team has 11 players in Soccer Class

```
import java.util.*;…

    // Write your overridden getNumberOfTeamMembers method here

}

public class Solution{

    public static void main(String []args){
        Sports c1 = new Sports();
        Soccer c2 = new Soccer();
        System.out.println(c1.getName());
        c1.getNumberOfTeamMembers();
        System.out.println(c2.getName());
        c2.getNumberOfTeamMembers();
    }
}
```

PRACTICE

Aim: Given an integer, n , perform the following conditional actions:

- If n is odd, print Weird
- If n is even and in the inclusive range of 2 to 5, print Not Weird
- If n is even and in the inclusive range of 6 to 20, print Weird
- If n is even and greater than 20, print Not Weird

Input Format

A single line containing a positive integer n.

Constraints

1 <= n <=100

Output Format

Print Weird if the number is weird; otherwise, print Not Weird.

Practical 15

- Ram is developing a restaurant management system using the following modules:

1. An interface RestDetails with a variable consisting of the restaurant name and a getMenu(String choice) method to take the user input for the different food items to be ordered.

2. Abstract class Order with an abstract function as getInvoice which will generate the total invoice amount and its discounted amount based on the food item’s ordered, if the amount exceeds 2000, apply discount of 20% and a constructor to take the user input for the food order such as sweet, spicy, sour, etc.

3. A class known as RestMgmt inherits the interface and Abstract Class to display the choices made, food items selected and the invoice generated

Practical 16

Sheetal visits the income tax website to find the taxable income from her gross income where the module takes as input the employee's gross salary and your total saving and uses another function named taxCalculate() to calculate your tax. The taxCalculate() function takes as parameters the gross salary as well as the total savings amount. The tax is calculated as follows:

(a) The savings is deducted from the gross income to calculate the taxable income. Maximum deduction of savings can be Rs. 1,00,000, even though the amount can be more than this.

(b) For up to 100,000 as taxable income the tax is 0 (Slab 0); beyond 100,000 to 200,000 tax is 10% of the difference above 100,000 (Slab 1); beyond 200,000 up to 500,000 the net tax is the tax calculated from Slab 0 and Slab 1 and then 20% of the taxable income exceeding 200,000 (Slab 2); if its more than 500,000, then the tax is tax from Slab 0, Slab 1, Slab 2 and 30% of the amount exceeding 500,000.

String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={‘j’, ‘a’, ‘v’, ‘a’, ‘t’, ‘e’, ‘c’, ‘h’ , ‘n’, ‘o’};
String s=new String(ch);
```

- is same as:

```
String s="javatechno";
```

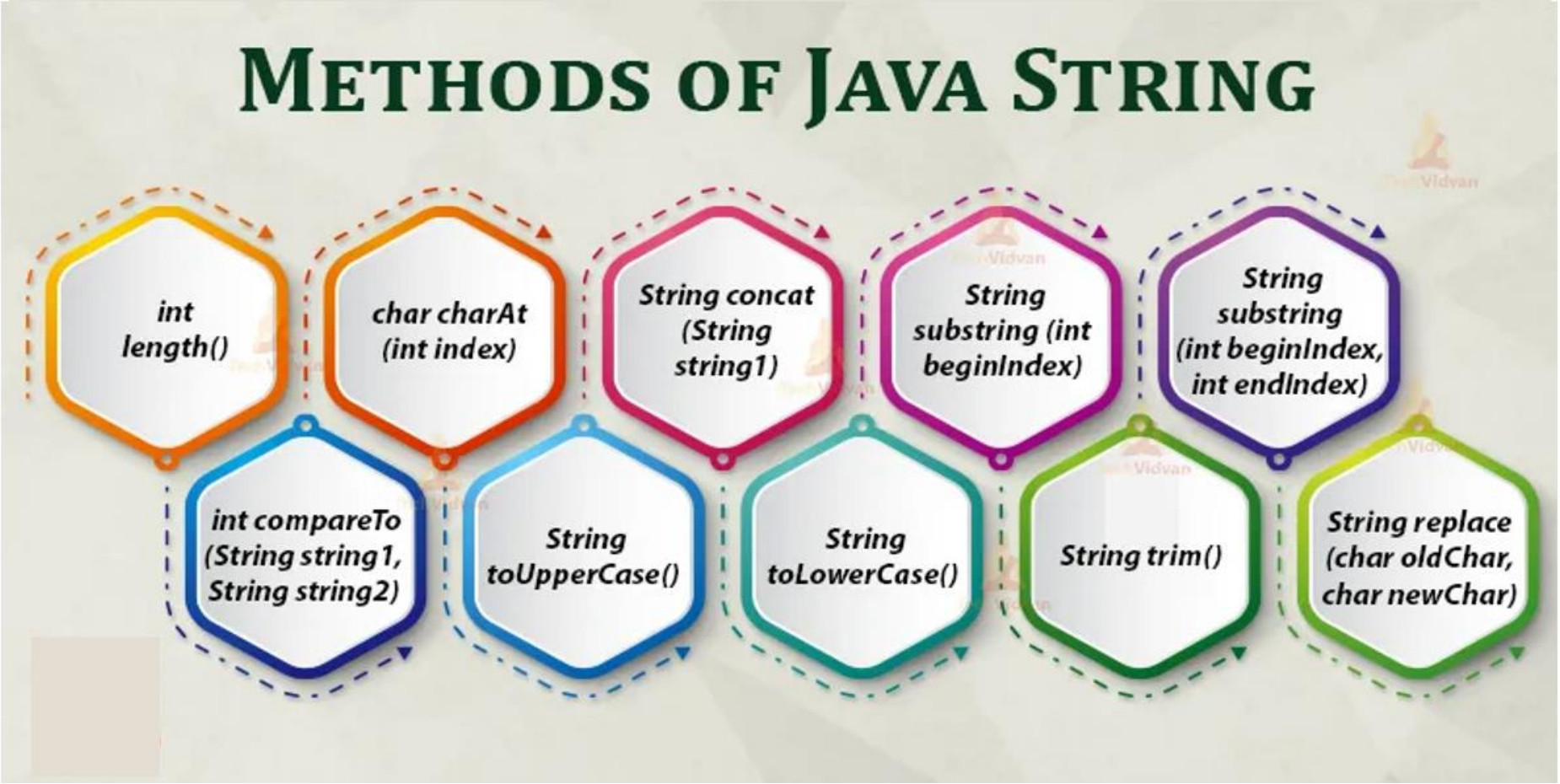
String

char charAt(int index)	It returns char value for the particular index
int length()	It returns string length
boolean equals(Object another)	It checks the equality of string with the given object.
boolean isEmpty()	It checks if string is empty.
String concat(String str)	It concatenates the specified string.
String replace(char old, char new)	It replaces all occurrences of the specified char value.
String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.

String

int indexOf(int ch)	It returns the specified char value index.
int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
int indexOf(String substring)	It returns the specified substring index.
int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
String toLowerCase()	It returns a string in lowercase.
String toUpperCase()	It returns a string in uppercase.

String


METHODS OF JAVA STRING
int length()
char charAt (int index)
String concat (String string1)
String substring (int beginIndex)
String substring (int beginIndex, int endIndex)
int compareTo (String string1, String string2)
String toUpperCase()
String toLowerCase()
String trim()
String replace (char oldChar, char newChar)

PRACTICE

Given two strings of lowercase English letters, X and Y, perform the following operations:

1. Sum the lengths of X and Y.

2. Determine if X is lexicographically larger than Y.

3. Capitalize the first letter in X and Y and print them on a single line, separated by a space.

Input Format

The first line contains a string . The second line contains another string . The strings are comprised of only lowercase English letters.

Output Format

Output - StringPrac (run)

```
run:
hi
students
10
No
Hi Students
BUILD SUCCESSFUL (total time: 10 seconds)
```

PRACTICE

```
package stringprac;
import java.util.*;
public class StringPrac
{
    public static void main(String[] args)
    {
    Scanner sc=new Scanner(System.in);
    String A=sc.next();
    String B=sc.next();
   System.out.println(A.length()+B.length());
   System.out.println(A.compareTo(B)>0?"Yes":"No");
   System.out.println(A.substring(0, 1).toUpperCase()+A.substring(1,A.length())+
                                                    “     “+B.substring(0,1).toUpperCase()+B.substring(1, B.length()));
    }
```

Output - StringPrac (run)

```
run:
hi
students
10
No
Hi Students
BUILD SUCCESSFUL (total time: 10 seconds)
```

PRACTICE

- Confuse your friends by jumbling two words. To avoid getting yourself into confusion follow a pattern to jumble the letters.
  Pattern to be followed is, pick a character from the first word and pick another character from the second word. Continue this process
- Take two strings as input , create a new string by picking a letter from string1 and then from string2, repeat this until both strings are finished and maintain the subsequence. If one of the strings is exhausted before the other, append the remaining letters from the other string all at once.

PRACTICE

```
import java.util.*;
public class SecretCodeCreation
{  public static void main(String[] args) {
  Scanner sc=new Scanner(System.in);
  String str1=sc.next();
  String str2=sc.next();
  String password="";
  int i,j;
 for(i=0,j=0;i<str1.length()&&j<str2.length();i++,j++)
 {
      password=password+str1.charAt(i)+str2.charAt(j);
 }
 if(i<str1.length())
   password=password+str1.substring(i,str1.length());
 if(j<str2.length())
  password=password+str2.substring(j,str2.length());
 System.out.println(password);
}}
```

PRACTICE

- **Problem Definition –** Rahul writes a research paper and copies in the paper from his peers. But he doesn’t want to be caught, so he changes words keeping the letter constant. That means he interchanges the positions of letters in words. You are the reviewer and you have to find if he has copied a certain word from the one adjacent peer who has submitted his research paper for the same conference, and give Rahul the markings he deserves.
- Note that: Uppercase and lowercase are the same.
- **Input Format:**
- First line with the adjacent student’s word
- Second line with Rahul’s word
- **Output Format:**
- 0 if not copied or 1 if copied

Date functions

- This object contain date and time components.
- *It is consist of year, month, day, hour, minute and second.*

```
import java.time.*;
LocalDate today = LocalDate.now();
System.out.println("Today's Date : "+ today);
```

Date functions

Prefix	Use
of	Creates an instance where the factory is primarily validating the input parameters, not converting them.
from	Converts the input parameters to an instance of the target class, which may involve losing information from the input.
parse	Parses the input string to produce an instance of the target class.
format	Uses the specified formatter to format the values in the temporal object to produce a string.
get	Returns a part of the state of the target object.
plus	Returns a copy of the target object with an amount of time added.
minus	Returns a copy of the target object with an amount of time subtracted.

Practical No. 17

A Birthday Reminder system is to be created with the following features:

a. Take the user input for name and date of birth. Consider the current month as October

b. In case of the candidate's birthday print the message 'Happy Birthday to You', else print the message 'Today is not my birthday'.

c. Also, calculate the age of the candidate(in terms of years, months and days)

Java Collections

- The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection **means a single unit of objects**. Java Collection framework provides many **interfaces (Set, List, Queue, Deque)** and **classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).**

Vectors

- **Java Vector class** comes under the **java.util** package.
- The vector class **implements a growable array of objects**. Like an array, it contains the component that can be **accessed using an integer index.**
- Vector is very useful if we don't know the size of an array in advance or we need one that can change the size over the lifetime of a program.
- Array is a set of similar data types which has **static memory allocation.**
- Vector **implements a dynamic array** that means it can grow or shrink as required. It is similar to the ArrayList, but with two differences-
  - Vector is **synchronized.**
  - The vector contains many legacy methods that are not the part of a collections framework
- The signature of the class is:

```
public class Vector<E>
extends Object<E>
implements List<E>, Cloneable, Serializable
```

Vectors(Contd.)

- **Constructors:**

Sr.No.	Constructor & Description
1	**Vector( )** This constructor creates a default vector, which has an initial size of 10.
2	**Vector(int size)** This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.
3	**Vector(int size, int incr)** This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.
4	**Vector(Collection c)** This constructor creates a vector that contains the elements of collection c.

Vectors(Contd.)

- **Example: Vector v1=new Vector();**
  - It creates an empty Vector with the default initial capacity of 10. It means the Vector will be re-sized when the 11th elements needs to be inserted into the Vector. Note: By default vector doubles its size. i.e. In this case the Vector size would remain 10 till 10 insertions and once we try to insert the 11th element It would become 20 (double of default capacity 10).
- **Example: Vector v2=new Vector(3);**
  - It will create a Vector of initial capacity of 3.
- **Example: Vector v3=new Vector(4,6);**
  - Here we have provided two arguments. The initial capacity is 4 and capacity increment is 6. It means upon insertion of 5th element the size would be 10 (4+6) and on 11th insertion it would be 16(10+6).

Vectors(Contd.)

- **void addElement(Object element):** It inserts the element at the end of the Vector.
- **void add(int index, Object element)**: Inserts the specified element at the specified position in this Vector.
- **int capacity():** This method returns the current capacity of the vector.
- **boolean contains(Object element):** This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
- **Object elementAt(int index):** It returns the element present at the specified location in Vector.
- **Object firstElement():** It is used for getting the first element of the vector.
- **Object lastElement():** Returns the last element of the vector.
- **boolean isEmpty():** This method returns true if Vector doesn’t have any element.
- **boolean removeElement(Object element):** Removes the specifed element from vector.

Vectors(Contd.)

- **boolean removeAllElements():** It Removes all those elements from vector and size becomes zero.
- **int size():** It returns the current size of the vector.
- **void setSize(int size):** It changes the existing size with the specified size.
- **boolean containsAll(Collection c):** It returns true if all the elements of collection c are present in the Vector.
- **Object elementAt(int index):** It returns the element present at the specified location in Vector.
- **Object get(int index):** Returns the element at the specified index.
- **void setElementAt(Object element, int index):** It updates the element of specifed index with the given element.

ArrayList

- The ArrayList class implements the List interface.

- It uses a **dynamic array to store the duplicate element of different data types**. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

- The ArrayList class of the Java collections framework provides the functionality of **resizable-arrays.**

- Before using ArrayList, we need to import the java.util.ArrayList package first. Here is how we can create arraylists in Java:

```
ArrayList<Type> arrayList= new ArrayList<Type>();
```

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

void add(int index, E element)
boolean add(E e)
boolean addAll(Collection<? extends E> c)
boolean addAll(int index, Collection<? extends E> c)
void clear()

int indexOf(Object o)
E remove(int index)
boolean remove(Object o)
boolean removeAll(Collection<?> c)
boolean removeIf(Predicate<? super E> filter)
protected void removeRange(int fromIndex, int toIndex)
void replaceAll(UnaryOperator<E> operator)
void retainAll(Collection<?> c)

ArrayList Functions

```
ArrayList<String> al=new ArrayList<String>();
al.add("Mango");
al.add("Apple");
al.add("Banana");
al.add("Grapes");

 //accessing the element
 System.out.println("Returning element: "+al.get(1));

//changing the element
al.set(1,"Dates");

System.out.println(fruit);

//Traversing list
for(String fruit:al)
        System.out.println(fruit);

// Remove element at index 3
a1.remove(3);
```

Output:

```
Returning element: Apple
[Mango, Dates, Banana, Grapes]
Mango
Dates
Banana
Grapes
```

Assigning values in ArrayList

```
String x[]={"1","2","3","4","5","6","7","8","9"};

ArrayList<Integer> alit=new ArrayList<Integer>(Arrays.asList(1,2,3,4,5,6,7,8,9,10,11));

ArrayList<String> alit2=new ArrayList<String>(Arrays.asList(x));

alit.size();     //To fetch the length
```

Iterator

An **Iterator is an object** that can be used to loop through collections, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping.

To use an Iterator, you must import it from the **java.util package**.

```
Iterator itr=list.iterator();
while(itr.hasNext())
{
System.out.println(itr.next());
}
```

boolean hasNext()
Returns true if the iteration has more elements. (In other words, returns true if next() would return an element rather than throwing an exception.)

next()
Returns the next element in the iteration.

ArrayList with Iterator

```
package collectionsprog;

import java.util.*;

public class CollectionsProg
{
  public static void main(String args[])
  {
      ArrayList<String> list=new ArrayList<String>();                    //Creating arraylist
      list.add("OOP");//Adding object in arraylist
      list.add("DBMS");
      list.add("CN");
      list.add("DS");
      //Traversing list through Iterator
      Iterator itr=list.iterator();
      while(itr.hasNext())
      {
      System.out.println(itr.next());
      }
   }
}
```

File Handling: File Functions

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

File Handling

```
package filehandling;

import java.io.File;
import java.util.Date;

public class Demo1_DirectoryList {
    public static void main(String args[]) {
        File file = new File("D:\\TEACHER");  //Displays all files & folders under mentioned directory
        String[] fileList = file.list();
        for(String name:fileList){
            System.out.println(name);
        }
    }
}
```

File Handling

Constructor and Description
`FileWriter(File file)` Constructs a FileWriter object given a File object.
`FileWriter(File file, boolean append)` Constructs a FileWriter object given a File object.
`FileWriter(FileDescriptor fd)` Constructs a FileWriter object associated with a file descriptor.
`FileWriter(String fileName)` Constructs a FileWriter object given a file name.
`FileWriter(String fileName, boolean append)` Constructs a FileWriter object given a file name with a boolean indicating whether or not to append the data written.

Practical No. 18

A file handler system within an Operating system corresponding to file creation and manipulation should consists of the following features:

a. Verify whether a particular file exists, if it does display its absolute path, otherwise create a new File.

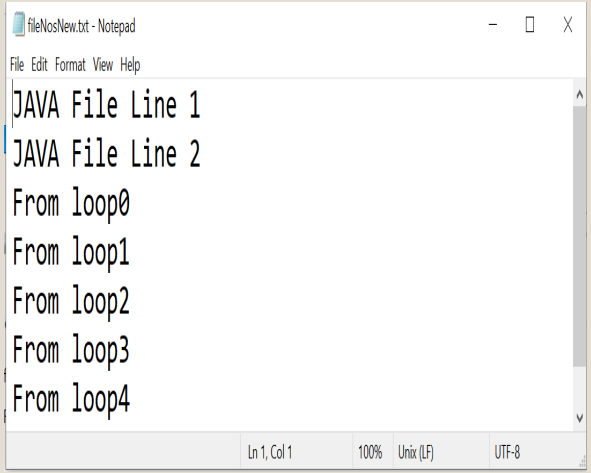
b. Input an array list within the file with names of students present within the session, along with printing the initial letter of your name.

File Handling

```
public static void main(String[] args) {
  try
  {
     File f=new File("D:\\javaprog\\DemoJ.txt");
     if(f.exists())
     {
        System.out.println("File exists");
        System.out.println(f.getAbsolutePath());
     }
     else
     {
        System.out.println("Does not exists");
        if(f.createNewFile())
        {
           System.out.println("File created");
        }
     }

 try (FileWriter fw = new FileWriter(f))
     {
        String s="This is Java Class. Students of Java Class
are"+System.lineSeparator();
        fw.write(s);
        ArrayList<String> al=new ArrayList<>();
        al.add("Ashka");
        al.add("Veer");
        al.add("Arjit");
        al.add("Ojas");
        al.add("Grishma");
        System.out.println(al);

        for(String x:al)
        {
           fw.write(x+System.lineSeparator());
        }
        fw.write(65+10);

        System.out.println("Done");
     }
  }
  catch(IOException e)
  {
     System.out.println(e.getMessage());
  }
}
```

Practical No 19: LINE DELETION using String Builder

Consider an efficient file handler system designed to write contents and read the same from a file, delete a specific statement based on the condition and it should display a message as '**Deleted**'. Incase of null value captured within the file, it should display an exception.



1. Write a file

2. Read the file using BufferedReader

3. if the string read is not null, append content

if content equals to a particular word, delete the line and continue

Line Deletion

```
package filehandling;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class LineDelete {
public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        String strLine = "";
        int i;
        try
          {
            String filename= "C:\\Users\\S340\\Desktop\\Ganpat-
OOP\\File\\fileNosNew.txt";

            try (FileWriter fw = new FileWriter(filename,false))
 {
              fw.write("JAVA File Line 1\n");
              fw.write("JAVA File Line 2\n");

             for (i=0; i<5; i++) {

                 fw.write("From loop"+i+"\n");

              }

           }
```

```
BufferedReader br = new BufferedReader(new
FileReader("C:\\Users\\S340\\Desktop\\Ganpat-OOP\\File\\fileNosNew.txt"));
            //read the file content
            String s = "From loop3";
            while (strLine != null)
            {
              sb.append(strLine);
              sb.append(System.lineSeparator());
             strLine = br.readLine();
             try {
             if (strLine.equals(s)){
                 strLine="deleted";}
             }
             catch (Exception e){
                 System.out.println("error->"+e);
             }System.out.println(strLine);
           }
           br.close();
          }
          catch(IOException ioe)
          {
           System.err.println("IOException: " + ioe.getMessage());
          }
    }
}
```

Line Deletion

fileNosNew.txt - Notepad

File Edit Format View Help

```
JAVA File Line 1
JAVA File Line 2
From loop0
From loop1
From loop2
From loop3
From loop4
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

```
JAVA File Line 1
JAVA File Line 2
From loop0
From loop1
From loop2
deleted
From loop4
```

Practical 20

- **Aim:** Anne was assigned to develop a text editor module which helps calculate the count of characters, words, lines, digits, white spaces and other characters in a given file. Input format: File with some alphanumeric contents, whitespaces and other symbols
- Output format: Count of characters, lines, characters, numbers, spaces and symbols if any.

myFile.txt - Notepad

File Edit Format View Help

```
I am Studying $JAVA $
$Java $ includes numbers 1234567890

Ganpat
ICT it is
```

```
run:
No. of character=47
No. of digit=10
No. of space=11
No. of words=11
No. of lines=10
No. of other characters=4
```

Practical 20 (Contd.)

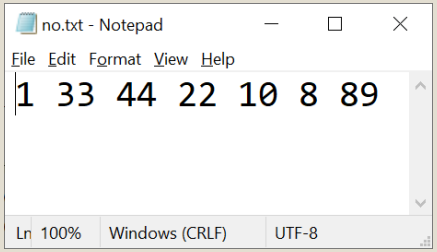
```
import java.io.*;
class Practical20
{
public static void main(String args[]) throws IOException
{
File f = new File("D://myFile.txt");
if(!f.exists())
{
System.out.print("File does not Exist");
}
BufferedReader fr=new BufferedReader(new FileReader(f));
int i, line=0,ch=0,digit=0,space=0,word=0,other=0;
String temp;
while((temp=fr.readLine())!=null)
{
    line++;
    for(i=0;i<temp.length();i++)
    {
    if(Character.isDigit(temp.charAt(i))){
        digit++;    }
   else if(Character.isLetter(temp.charAt(i))){
        ch++;       }
    else if(Character.isSpace(temp.charAt(i))){
        space++;
        word++;
    }
    else{
        other++;
    }
  }
}
```

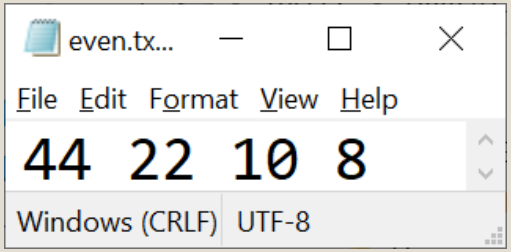
Practical 20 (Contd.)

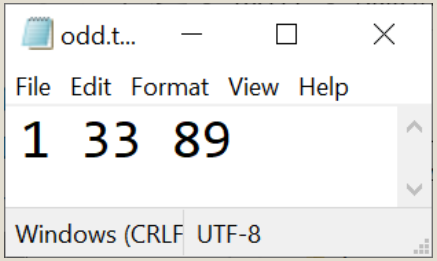
```
  System.out.println("No. of character="+ch);
  System.out.println("No. of digit="+digit);
  System.out.println("No. of space="+space);
  System.out.println("No. of words="+word);
  System.out.println("No. of lines="+line);
  System.out.println("No. of other characters="+other);
   fr.close();
 }
}
```

Practical 21

Aim: Athia was assigned a task to separate the odd numbers and even numbers from the numbers read collectively from a file. Develop the module in such a way that the storage of odd numbers and even numbers takes place respectively.







Practical 21 (Contd.)

```
import java.io.*;
public class Practical21
{
public static void main(String args[])throws IOException
{
FileWriter f1=new FileWriter("D:\\odd.txt");
FileWriter f2=new FileWriter("D:\\even.txt");
FileReader fs = new FileReader("D:\\no.txt");
                    BufferedReader br = new BufferedReader(fs);
                    String temp = "";
                    while((temp=br.readLine())!=null)
{
String[] words=temp.split("\\s");     //Split based on whitespace
for(String w:words)
{
```

```
if(Integer.parseInt(w)%2==0)
{
f2.write(w+" ");
System.out.println("Even No.: "+w+" ");
}
else
{
f1.write(w+" ");
System.out.println("Odd No.: "+w+" ");
}
}
}
f1.close();
f2.close();
}
}
```

Practice

Aim: Write a program to read from a file up to n lines. The lines are read entirely and displayed on the console. The program also counts lines beginning with character ‘A’ or ‘E’ as first letter and displays the count on the console and in a new file created programmatically. If the word **quit (irrespective of upper case or lower case)** is encountered within the file being read, print the message ‘Data terminated’ on the console.

myFile.txt - Notepad

File Edit Format View Help

```
Am Studying $JAVA $
$Java $ includes numbers 1234567890

Ganpat Elegant
A insititute
ICT it is

quit
```

```
run:
Am Studying $JAVA $
$Java $ includes numbers 1234567890

Ganpat Elegant
A insititute
ICT it is

quit
Data Terminated
No of lines start with A or E= 2
```

Ab.txt - Notepad

File Edit Format View Help

```
No of lines start with A or E= 2
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Practice

```
public static void main(String args[]) throws IOException {
BufferedReader br = new BufferedReader(new FileReader("D:\\myFile.txt"));
FileWriter fw=new FileWriter("D:\\Ab.txt");
int count=0;
String str="";
while((str=br.readLine())!=null) {
System.out.println(str);
if(str.startsWith("A") || str.startsWith("E"))
{
count++;
}
if(str.equalsIgnoreCase("Quit"))
{
   System.out.println("Data Terminated");
} }
System.out.println("No of lines start with A or E= "+count);
fw.write("No of lines start with A or E= "+count);
fw.close();
br.close();  }
```

Practical No 22

Aim: Anshul was provided with a task to optimize the system by implementing multithreading. Consider three threads Thread1, Thread2 and Thread3, Thread2 should run followed by Thread1 set the priorities accordingly. Thread 3 should execute only after Thread2 execution is completed. Also display the meta data of threads such as id, name, alive status.

```
Thread Running
Id is: 15
Thread Running
Id is: 14
Priority  for Thread-1 is: 10
Thread is currently active
Thread is currently active
Thread is currently active
Priority  for Thread-0 is: 1
Thread is currently active
Thread is currently active
Thread is currently active
Thread Running
Id is: 16
Priority  for Thread-2 is: 5
Thread is currently active
Thread is currently active
Thread is currently active
Thread Alive: Thread1: false Thread2: false Thread3: true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical No 22

```
package multithreading;

class Multithreading extends Thread
{
 public void run()
{
   System.out.println("Thread Running \nId is: "+Thread.currentThread().getId());
   System.out.println("Priority  for "+Thread.currentThread().getName()+" is: "+Thread.currentThread().getPriority());
   for(int i=0;i<3;i++)
   {
      System.out.println("Thread is currently active");
   }
}
public static void main(String args[])
{
Multithreading t1=new Multithreading();
Multithreading t2=new Multithreading();
Multithreading t3=new Multithreading();
t1.start();
t2.start();

t1.setPriority(Thread.MIN_PRIORITY);
t2.setPriority(Thread.MAX_PRIORITY);

try{
   t2.join();
}
catch(Exception e){}
t3.start();
System.out.println("Thread Alive: Thread1: "+t1.isAlive()+" Thread2: "+t2.isAlive()+" Thread3: "+t3.isAlive());
}
}
```

Practical No 23

Aim: Ayesha was assigned a task to store the records of an employee with information such as employee id, name, age and salary in a table. Using the concept of JDBC, create a table Employee which consists of the columns as mentioned and store the data collected through user input within the table.

Practical No 23

```
package databasehandling3
import java.util.*;
import java.sql.*;
public class InsertData {
public static void main(String args[]) throws ClassNotFoundException, SQLException
{
        Scanner sc=new Scanner(System.in);
        String ename;
        int eid, age, salary;
    try{
                                Class.forName("com.mysql.jdbc.Driver"); //driver
                                String driverUrl = "jdbc:mysql://localhost:3306/demo";
                                Connection con = DriverManager.getConnection(driverUrl,"root","ganpat"); //connection
                                String createTableQuery ="CREATE table EmployeeTb( "
                                                + "id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT , "
                                                + "name VARCHAR(30) ,"
                                                + "age INTEGER(12), salary INTEGER(10))";
```

Practical No 23

```
PreparedStatement CreateTable=con.prepareStatement(createTableQuery); //prepared statement for create table
                                        if(CreateTable.execute())
                    {
                    System.out.println(" Table created successfully ");
                    }
//insert query
                                        String InsertDataQuery = "INSERT INTO EmployeeTb(ename,age,salary) VALUES(?,?,?)";

                    //getting values form the user
                    System.out.println("Enter employee id : ");
                                        eid = sc.nextInt();
                                        System.out.println("Enter employee name : ");
                                        ename = sc.next();
                    System.out.println("Enter employee age : ");
                                        age = sc.nextInt();
                                        System.out.println("Enter employee salary : ");
                                        salary = sc.nextInt();
```

Practical No 23

```
//prepared statement for insert query
        PreparedStatement InsertData = con.prepareStatement(InsertDataQuery);
                //applying setstring to add values to insert query
                        InsertData.setInt(1,eid);
                        InsertData.setString(2,ename);
                          InsertData.setInt(3,age);
                         InsertData.setInt(4,salary);
                        //for applying changes to the table
                        InsertData.executeUpdate();
                        //close the connection and query satement
                        InsertData.close();
                        con.close();
}
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Practice

Aim: Ayesha was assigned a task to store the records of an employee with information such as employee id, name, age and salary in a table. Using the concept of JDBC, create a table Employee which consists of the columns as mentioned and store the data collected through user input within the table.

In extension to the above program, fetch the details of employees whose age is greater than 25.