JAVA
Chapter-2

What is an Array? Explain types of arrays in java?
An array is a set of consecutive memory locations having same name and sametype of data. Java provides three types of arrays as follows.

1. Single Dimensional Array: -
An array with only one sub-script is known as a single dimensional array. It is used to represent a list.
Syntax: -
data_type array_name[] = new data_type[size];
Example: -
int a[] = new int[10];
This means "a" is a single dimensional array used to store 10 integers.

2. Two Dimensional Array (or) Double Dimensional Array: -
An array with two sub-scripts is known as a double dimensional array. It isused to represent a table or a matrix.
Syntax: -
data_type array_name [][] = new data_type[rows][columns];
Example: -
int m[][]=new int[4][6];
This means "m" is a double dimensional array used to store 24 integers in 4 rows and 6 columns.

3. Multi-Dimensional Array: -
An array with more than two sub-scripts is known as a multi-dimensional array.
Syntax: -
data_type array_name [][][]=new data_type[rows][cols][tables];
Example:-
int p[][][]=new int[4][6][3];
This means "p" contains 3 tables each table contains 4 rows and 6 columns.

Accessing array elements using for each loop: -
For each loop is used access array elements from first index to last index without explicitly specifying its positions.
Syntax: -
```
    for (data_type variable : array_name)
    {
       Body of the loop;
    }
```
/* Java program to access values of an array using for each loop */
```
import  java.io.*;
class StringArray
{
  public static void main(String  args[])
  {
   String[] arr = {"Anji", "Satya", "Tulasi" , "Lakshmi"};
   System.out.println("Accessing array elements using subscript");
   for (int i=0; i<arr.length; i++)
   {
       System.out.println(arr[i]);
   }
   System.out.println("Accessing array elements using for each loop");
   for(String x : arr)
   {
      System.out.println(x);
   }
  }
}
```
Explain Command Line Arguments in Java?
command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received by the main function as a string and it can be used as an input. So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) number of arguments from the command prompt.

Simple example of command-line argument in java

```
class Main
{
  public static void main(String args[])
  {
    System.out.println("Your first argument is: " + args[0]);
  }
}
```

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

How to Compile: -
```
javac Main.java
```

How to Run: -
```
java Main Ramachnadra
```
Output: - Your first argument is: Ramachandra

Another example of command-line argument in java
In this example, we are trying to pass more arguments at the command-line.
To access the values, we have to traverse the array using for loop.

```
/* Write java program to read student details using command line
arguments */
class Main
{
   public static void main(String args[])
   {
     for(int i=0;i<args.length;i++)
     {
        System.out.println(args[i]);
     }
   }
}
```
How to Compile: -
```
Javac Main.java
```
How to Run: -
```
Java Main Rajani B.Sc 26-10-2001 45000.00
```
Output: -
```
Rajani
B.Sc
26-10-2001
45000.00
```

What is inheritance? Explain types of inheritance in Java?

Inheritance: The process of acquiring the properties of one object by another object is called inheritance. (or) The mechanism of deriving a new class from old class is called inheritance.
The old class is called as super class/ Base class/ parent class.
The new class is called as derived class/ sub-class/ child class.
The inheritance allows sub classes to inherit, all variables and methods of their parent class.
The Inheritance concept is used to reuse the variables and methods of a class in its subclass. So inheritance allows reusability of class members.

SYNTAX: -
```
class subclass_name extends superclass_name
{
     Variable declaration;
     Method declaration;
}
```

- In the above syntax subclass name and superclass name are valid identifiers.
- **extends** is a keyword, and it is used to extend the properties of superclass to the sub class.
- The subclass will contain its own variables and methods as well as variables and methods of superclass.

Types of inheritance: -
Inheritance is classified into 5 types.
1. Single inheritance.
2. Multiple-inheritance.
3. Multi-level inheritance
4. Hierarchical inheritance.
5. Hybrid Inheritance.

1. Single inheritance: -
The process of deriving only one sub class from one super class is called single inheritance. In the single Inheritance only one pair of Super class and one sub class exists.

$$\begin{tikzcd}\textbf{[A]}\ \text{(Super class)} \arrow[d] \\ \textbf{[B]}\ \text{(Sub class)}\end{tikzcd}$$

Syntax: -
```
class superclass name
{
      Variable declaration;
      Method declaration;
}
class subclassname extends superclassname
{
     Variable declaration;
     Method declaration;
}
```
Example: -
```
class A
{
     Variable declaration;
     Method declaration;
}
class B extends A
{
     Variable declaration;
     Method declaration;
}
```
Important Rule: -
Parent class object can access methods and members of parent class only. But sub class object can access methods and members of both sub class and parent class.
```
/* Java program using single inheritance */
import java.util.Scanner;
class A
{
  int a;
  A()
  {
      a=5;
  }
  void showa()
  {
     System.out.println("a="+a);
  }
}
class B extends A
{
  int b;
  B()
  {
      b=10;
  }
  void showb()
  {
     System.out.println("b="+b);
  }
  void sum()
  {
      System.out.println("Sum="+(a+b));
   }
}
class  Single
{
   public static void main(String args[])
   {
      B x = new B();
      x.showa();
      x.showb();
   }
 }
```

2. Multiple Inheritance: -
The process of deriving the new class from more than one old class is known as multiple inheritance.

$$\begin{tikzcd} \textbf{Super\_class1 [A]} \arrow[dr] & & \textbf{[B] Super\_class2} \arrow[dl] \\ & \textbf{[C] Sub\_Class} & \end{tikzcd}$$

Java does not support multiple-inheritance. But it can be implemented by using interface.

What is an interface, Explain?

Interface is used to define standard behavior that can be implemented by a class or anywhere in the class hierarchy. An interface contains final variables (constant values) and abstract methods. The difference between class and interface is that the methods in an interface are only declared but not implemented, that means the methods do not have body. The methods in an interface should be public.

Syntax to declare an interface: -

```
public interface <interface-name>
{
 interface body;
}
```

Differences between class and interface: -

Class	**Interface**
In class, we can instantiate variables and create an object.	In an interface, we can't instantiate variables and we can't create an object.
Class can contain concrete methods (methods with implementation)	The interface cannot contain concrete methods
The access specifiers used with class are private, protected and public.	The access specifiers used with Interface is public only.
To define subclass the keyword extends is used	To define subclass the keyword implements is used
Class is used for data encapsulation	Interface is used for data abstraction
We can't achieve multiple inheritance using class	Interface allows to achieve multiple inheritance in java

/* Write a program to implement multiple inheritance */

```
import java.io.*;
interface Home
{
   String fn="Srinivas";
   String mn="Swathi";
   String a="Kakinada";
   public void bioData();
}
interface College
{
   String cn="Sri Chaitanya College";
   String pn="Mohan Rao";
   String ca="Vizag";
   public void academicData();
}
class Student implements Home, College
{
   String sn,c;
   double  f;
   int  pm;
   void input()
   {
      sn = "Chinmayi";
      c = "MCA";
      f = 20000;
      pm = 98;
   }
  public void bioData()
  {
     System.out.println("\nStudent Name   : " + sn);
     System.out.println("\nFather Name    : " + fn);
     System.out.println("\nMother Name    : " + mn);
     System.out.println("\nResidential Address   : " + a);

  }
  public void academicData()
  {
     System.out.println("\nStudent Course   : " + c);
     System.out.println("\nCollege Name   : " + cn);
     System.out.println("\nPrincipal Name   : " + pn);
     System.out.println("\nCollege Address  : " + ca);
     System.out.println("\nCourse Fee     : " + f);
     System.out.println("\nPercentage of Marks   : " + pm);
  }
}
class Main
{
  public static void main(String x[])
  {
    Student k = new Student();
    k.input();
    k.bioData();
    k.academicData();
  }
}
```

/* java program to implement multiple inheritance */

```
import java.io.*;
interface A
{
  public void m1();
  public void m2();
}
interface B
{
  public void m3();
  public void m4();
  public static int t1=20, t2=50;
}
class K implements A,B
{
   public void m1()
   {
     System.out.println("It is my first method");
   }
   public void m2()
   {
     System.out.println("It is my second method");
   }
   public void m3()
   {
     System.out.println("It is my third method");
   }
   public void m4()
   {
     System.out.println("It is my fourth method");
   }
   void m5()
   {
     System.out.println("It is my fifth method");
     System.out.println("First Term Rank = " + t1);
     System.out.println("Second Term Rank = " + t2);
   }
}
class  Myclass
{
   public static void main(String args[])
   {
     K x = new K();
     x.m1();
     x.m2();
     x.m3();
     x.m4();
     x.m5();
    }
}
```

/* Write java program with real time example of an interface */

```
interface Couriers
{
    public void  time();
    public void  cost();
    public void  service();
}
class Professional implements Couriers
{
   public void time()
   {
      System.out.println("Minimum time 4 hours ");
   }
   public void cost()
   {
      System.out.println("Minimum cost Rs.50/- ");
   }
   public void service()
   {
      System.out.println("It is an international courier service");
   }
   void contact()
   {
      System.out.println("Local contact person : Mr. P S Prasd");
   }
}
class DTDC implements Couriers
{
   public void time()
   {
      System.out.println("Minimum time 12 hours");
      System.out.println("Maximum time depending on the distance ");
   }
   public void cost()
   {
      System.out.println("Minimum cost Rs.30/- ");
   }
   public void service()
   {
      System.out.println("It is the most secured national courier");
   }
   void booking()
   {
      System.out.println("You can book through Online or Offline");
   }
}
class ANL implements Couriers
{
   public void time()
   {
      System.out.println("Minimum time 1 day");
   }
   public void cost()
   {
      System.out.println("Minimum cost Rs.20/- ");
   }
   public void service()
   {
      System.out.println("It is a popular domestic service");
   }
   void transport()
   {
      System.out.println("Way of transport: Using RTC and Private");
   }
}
class MyExample
{
   public static void main(String args[])
   {
      Professional x = new Professional();
      DTDC y = new DTDC();
      ANL z = new ANL();
      System.out.println("About Professional courier");
      x.time();
      x.cost();
      x.service();
      x.contact();
      System.out.println("\n\nAbout DTDC courier");
      y.time();
      y.cost();
      y.service();
      y.booking();
      System.out.println("\n\nAbout ANL courier");
      z.time();
      z.cost();
      z.service();
      z.transport();
   }
}
```

Types of inheritance: -

3. Multilevel Inheritance: -

If a sub class is derived from another sub class then it is called multilevel inheritance.

$$\begin{tikzcd}
\textbf{[A] Super class} \arrow[d] \\
\textbf{[B] Intermediate Base class} \arrow[d] \\
\textbf{[C] Sub class}
\end{tikzcd}$$

Syntax: -

```
class A
{
     Variable Declaration;
     Method declaration;
}
```

```
class B extends A
{
     Variable declaration
     Method declaration;
}
class C extends B
{
     Variable declaration;
     Method declaration;
}
```

In the above syntax **A, B** and **C** are valid identifiers. Class **C** contains its own variables and methods as well as the variables and methods of both the classes **A** and **B.**

4. Hierarchical inheritance: -
The process of deriving more than one class from an old class is called hierarchical inheritance.

$$\begin{tikzcd}
 & {[A]} \arrow[dl] \arrow[d] \arrow[dr] & \\
{[B]} & {[C]} & {[D]}
\end{tikzcd}$$

Syntax: -

```
class A
{
     Variable declaration;
     Method declaration;
}
class B extends A
{
     Variable declaration;
     Method declaration;
}
class C extends A
{
     Variable declaration;
     Method declaration;
}
```

5. Hybrid inheritance: -
The combination of two or more types of inheritances is known as hybrid inheritance.The purpose of using hybrid inheritance in Java is to modularize the code base into well-defined classes and provide code reusability. For example in the following diagram we combine Hierarchical and multiple inheritances to create hybrid inheritance.

$$\begin{tikzcd}
 & {[A]} \arrow[dl] \arrow[dr] & \\
{[B]} \arrow[dr] & & {[C]} \arrow[dl] \\
 & {[D]} &
\end{tikzcd}$$

```
Interface A
{
     Constant Variables;
     Abstract Methods;
}
Interface B extends A
{
     Constant Variables;
     Abstract Methods;
}
Interface C extends A
{
     Constant Variables;
     Abstract Methods;
}
class D implements A, B
{
     Variable declaration;
     Method declaration;
}
```

/* Write java program using hybrid inheritance */

```
interface A
{
  int a=10;
  public void display_a();
}
interface B extends A
{
  int b=20;
  public void display_b();
}
interface C extends A
{
  int c=30;
  public void display_c();
}
class D implements B, C
{
  int sum,max;
  public void proceed()
  {
    sum=a+b+c;
    if(a>b && a>c)
      max=a;
    else if(b>c)
      max=b;
    else
      max=c;
  }
  public void display_a()
  {
    System.out.println("a=" + a);
  }
  public void display_b()
  {
    System.out.println("b="+b);
  }
  public void display_c()
  {
    System.out.println("c="+c);
  }
  public void display_all()
  {
    System.out.println("sum = " + sum);
    System.out.println("maximum = " + max);
  }

}
class Main
{
  public static void main(String args[])
  {
    D obj = new D();
    obj.proceed();
    obj.display_a();
    obj.display_b();
    obj.display_c();
    obj.display_all();
  }
}
```

Output: -

```
a=10
b=20
```

```
c=30
sum = 60
maximum = 30
```

Explain about "*this*" keyword in Java?

In Java, the keyword "this" is used to refer the current object. It is used to access instance variables and methods in the current class. Ex1: - If there is ambiguity between the instance variables and parameters in a method, then this keyword resolves the problem of ambiguity.
It is explained in the following example.

```
Import java.io.*;
class Student
{
    String name;
    double fee;
    void input(String name, double fee)
    {
       this.name = name;
       this.fee = fee;
    }
    void output()
    {
      System.out.println("Name = " + name);
      System.out.println("Fee = " + fee);
    }
}
class This1
{
    public static void main(String args[])
    {
     Student x = new Student();
     x.input("Srinivas",75000);
     x.output();
    }
}
```

**/* this() method call can be used to invoke current class constructor.
It is explained in the following program */**

```
import java.io.*;
class Student
{
 int id;
 String name;
 Student()
 {
   System.out.println("Default constructor is invoked");
 }
 Student(int id,String name)
 {
   this();    // To invoke default constructor
   this.id = id;
   this.name = name;
 }
 void display()
 {
   System.out.println("Student ID = " + id);
   System.out.println("Student Name = " + name);
 }
}
class This2
{
 public static void main(String args[])
 {
   Student x = new Student(111,"kiran");
   Student y = new Student(222,"Charan");
   x.display();
   y.display();
 }
}
```

Output: -

```
Default constructor is invoked
Default constructor is invoked

Student ID = 111
Student Name = Kiran

Student ID = 222
Student Name = Charan
```

Explain about "super" keyword in Java?

The super keyword in java is a reference variable that is used to refer immediate parent class object.

Usage of super Keyword: -

1. super is used to access immediate parent class instance variable.
2. super is used to invoke immediate parent class method.
3. super() is used to invoke immediate parent class constructor.

```
/* Write Java program to accept super class members using super
keyword */
import java.io.*;
class Vehicle
{
   int speed=50;
}
class Bike extends Vehicle
{
 int
 speed=100;
 void
 display()
 {
   System.out.println("Bike Speed = " + speed);
   System.out.println("Vehicle Speed = " + super.speed);
 }
}
class Super1
{
   public static void main(String args[])
   {
       Bike  x = new
       Bike();
       x.display();
   }
}
```

```
/* Write java program to invoke immediate parent class method using
super */
import java.io.*;
class A
{
  int  a=10;
  void display()
  {
     System.out.println("Value of class A = " + a);
  }
}
class B extends A
{
  int  b=20;
  void display()
  {
     super.display();
     System.out.println("Value of class B = " + b);
  }
}
class C extends B
{
  int  c=30;
  void display()
  {
     super.display();
     System.out.println("Value of class C = " + c);
  }
}
class Super2
{
   public static void main(String args[])
   {
      C obj = new C();
      obj.display();
   }
}
```

Output: -

```
Value of class A = 10
Value of class B = 20
Value of class C = 30
```

/* Java program to invoke parent class constructor using super() method */

```
import java.io.*;
class Vehicle
{
     Vehicle(String vc)
     {
         System.out.println("Vehicle colour is " + vc);
      }
}
class Bike extends Vehicle
{
     Bike(String vc, String bc)
     {
          super(vc);
          System.out.println("Bike colour is " + bc);
     }
}
class  Super3
{
     public static void main(String args[])
     {
          Bike b=new Bike("Red","Black");
     }
}
```

Output: -

```
Vehicle colour is Red
Bike colour is Black
```

Explain about "final" keyword in Java?

in Java, the keyword "final" is used to declare instance variable. If you make any instance variable as final, you cannot change its value. It will be regarded as a constant.

/* Write java program using final variable */

```
import java.io.*;
class College
{
     final String cn="Aditya, Kakinada";
     long p;
     double f;
     void input()
     {
       cn="KIET, Korangi"; //Error because "cn" is final
       variable p=245165;
       f=45000.00;
     }
     void output()
     {
       System.out.println("College Name : " + cn);
       System.out.println("Phone Number : " + p);
       System.out.println("College Fee : " + f);
```

```
    }
}
class Final1
{
   public static void main(String args[])
   {
     College x = new
     College(); x.input();
     x.output();
  }
}
```

FINAL METHOD

A method which is declared using the keyword "final" is known as a "final method". If you make any method as final, you cannot override it.

/* Write java program to define a final method. */

```
import java.io.*;
class Human
{
  final void food()
  {
    System.out.println("Human eat raw and cooked food");
  }
}
class Baby
{
   void food()  //Error because final method cannot be Overridden
   {
     System.out.println("Baby eats Cerelac or Nestum");
   }
}
class Final2
{
   public static void main(String args[])
   {
     Baby x = new Baby();
     x.food();
   }
}
```

FINAL CLASS

If you make any class as final, you cannot extend it. That means a final class should not contain any sub class.

/* Write java program to create a final class */

```
import java.io.*;
final class Bike
{
   Bike()
   {
     System.out.println("This is bike class");
   }
}
class HeroHonda extends Bike
{
  void output()
  {
     System.out.println("Hero Honda, is an Indian multinational
motorcycle and scooter manufacturer");
  }
}
class Final3
{
  public static void main(String args[])
  {
    HeroHonda x = new HeroHonda();
    x.output();
  }
}
```

Output: - Error because we cannot inherit HeroHonda from final Bike

What is Method Overriding? Explain its features?

If a child class has the same method which is already defined in its parent class, then it is known as method overriding in Java.

Rules for Java Method Overriding

1. The method must have the same signature as in the parent class. The means the overloaded method should have same name, same parameters and same return type.
2. There must be an IS-A relationship between the classes.

Usage of Java Method Overriding

1. Method overriding is used to provide specific implementation for child class method which is already defined by its parent class.
2. Method overriding is used for runtime polymorphism

Note: -

1. A static method cannot be overridden. Because static method is bound with class whereas instance method (overridden method) is bound with object. Static belongs to class area and instance belongs to heap area.
2. We cannot override main() method because is main() is a static method.

/* Write java program using method overriding */

```
import  java.io.*;
class Bank
{
    int getRateOfInterest()
    {
        return 8;
    }
}
class SBI extends Bank
{
    void aboutus()
    {
       System.out.println("SBI is a Public Sector Bank");
    }
}
class ICICI extends Bank
{
    int getRateOfInterest()
    {
        return 9;
    }
}
class AXIS extends Bank
{
    int getRateOfInterest()
    {
        return 10;
    }
}
class Baroda extends Bank
{
}
class Main
{
 public static void main(String args[])
 {
  SBI s = new SBI();
  ICICI i = new ICICI();
  AXIS a = new AXIS();
  Baroda b = new Baroda();
  System.out.println("SBI Interest = "+ s.getRateOfInterest());
  System.out.println("ICICI Bank Interest = " + i.getRateOfInterest());
  System.out.println("AXIS Bank Interest = " + a.getRateOfInterest());
  System.out.println("Baroda Bank Interest =" + b.getRateOfInterest());
 }
}
```

Output: -

```
SBI Interest = 8
```

```
ICICI Bank Interest = 9
AXIS Bank Interest = 10
Baroda Bank Interest = 8
```

Discuss about packages in java?

Definition: - A package is an encapsulation mechanism to group similar or related classes and interfaces as a single unit. A package is a group of similar type of classes, interfaces, exceptions and errors.

Advantages: -

Package provides the following features.

1. Naming conflicts were resolved.
2. It is possible to identify every java component uniquely.
3. Modularity of an application will be improved.
4. Maintainability of an application will be improved
5. Security for classes.

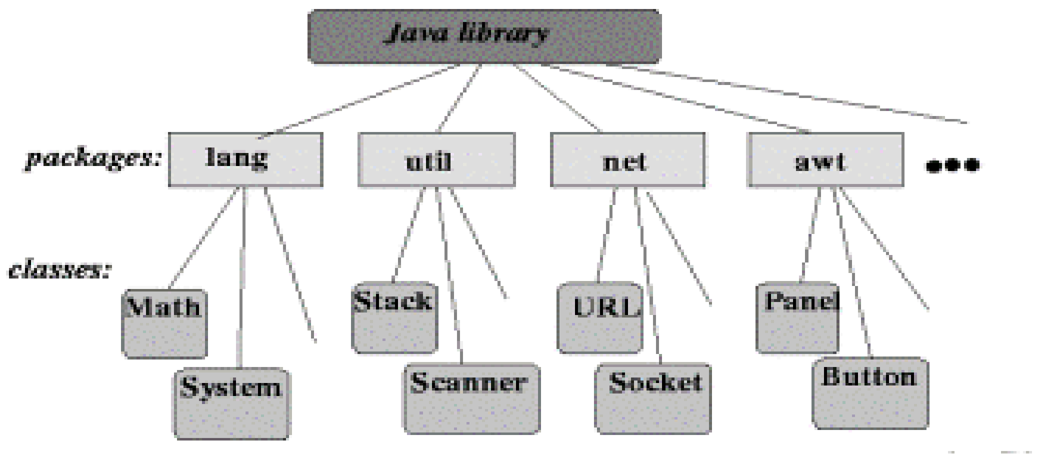
Types of Packages: -

Java provides two types of packages. They are …

i) Predefined packages (or) Built-in packages

ii) User-Defined packages.

Predefined Packages: - There are many predefined packages in java such as lang, net, io, util, sql, awt, javax, swing, etc. Built-in packages consist of a large number of classes which are a part of Java API (application programming interfaces). Some of the commonly used built-in packages are explained here under.



1. **java.lang: -** It contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported. The classes available in Java.lang package are Class, ClassLoader , Compiler, Loader, Math, Number , Byte, Double, Float, Integer, Boolean, Character, Long, Short, String, StrictMath, , StringBuffer, System, Thread, Throwable, etc.
2. **java.io: -** It provides classes to perform Input output operations. The classes available in Java.io package are FileInputStream, FileOutputStream, BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, DataInputStream, DataOutputStream etc.
3. **java.util: -** It provides utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc. The classes available in Java.util package are Stack, Scanner, Timer, Random, Calendar, Currency, Dictionary, EnumMap, HashMap, ArrayList, etc.
4. **java.applet: -** It is the smallest package in java. It provides classes necessary to create an applet. An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.
5. **java.awt: -** It provides classes to develop any GUI application. The classes available in java.awt package are TextField, TextArea, Label, RadioButton, CheckBox, Choice, List, Font, Frame, Graphics, etc.
6. **java.net: -** It provides classes to perform networking operations. The classes available in java.net package are URL, URLConnection,

HttpURLConnection, DatagramPacket, DatagramSocket, InetAddress, DatagramSocketImpl, Socket, MulticastSocket, ServerSocket, etc.

```
/* Write java program using predefined package */
import java.lang.String;
import java.lang.Math;
class PackExample
{
  public static void main(String args[])
  {
    String obj = new String("Aditya");
    System.out.println("String Length = " + obj.length());
    System.out.println("String in lower case = " + obj.toLowerCase());
    System.out.println("String in upper case = " + obj.toUpperCase());
    System.out.println("Square root of 121 =  " + Math.sqrt(121));
  }
}
```

USER DEFINED PACKAGE

We can create user defined package using the following syntax.
package package_name;
Ex: - package college;

Main Points: -

1) A package name may be any valid identifier and it should be starts with a small letter.
2) To create a package, package statement must be the first non-comment statement in the program. After that import statement is allowed.
3) In any java source file at most one package statement will be allowed. That means we can create only one class in one package at a time. To create another class for the same or different package we need to write separate program.
4) Save the program with class_name and use ".java" as its extension.
5) A package should contain public classes only.
6) Use the following command to compile the package program.

```
javac  -d . filename.java
```

Here "d" stands for destination, and "." denotes present working directory. (current directory)

7) If the package class contains main() method, run the program using the following syntax.

```
java packagename.classname
```

8) To import a class from a user defined package, the following syntax is used.

```
import  packagename.classname;
```

Example1: - Write and execute package demo program.

```
package Mypack1;
public class Main
{
   public static void main(String[] args)
   {
     System.out.println("Package Demo Program");
   }
}
```

```
C:\documents>  javac –d . Main.java
```

Result: As a result, a new package "Mypack1" is created in which "Main.class" is placed.

```
C:\documents>  java Mypack1.Main
```

Output: Package Demo Program

Example2: - Create user defined package Message with two classes and access the methods from another program by importing those classes._

Step1: - Creating College class in message package

```
package Message;
```

```
public  class  College
{
  public void msg1()
  {
    System.out.println("Tomorrow is holiday due to Sunday");
  }
  public void  msg2()
  {
    System.out.println("Last date to pay I Term fee is 10-10-2022!");
  }
  public void  msg3()
  {
    System.out.println("Today we have a seminar at 3PM");
  }
}
```

Compile the program using the following command. Don't Run the program

```
javac  -d  .  College.java
```

Step2: - Creating Mobile class in message package

```
package Message;
public  class  Mobile
{
  public void  alert1()
  {
    System.out.println("your validity will be expired within two days");
  }
  public void  alert2()
  {
    System.out.println("Recharge immediately for uninterrupted calls");
  }
  public void  recharge()
  {
    System.out.println("Recharge with 75/- for 21 days unlimited calls");
  }
}
```

Compile the program using the following command. Don't Run the program

```
javac  -d  .  Mobile.java
```

Step3: - accessing College and Mobile classes from message package

```
import Message.College;
import Message.Mobile;
class  Student
{
   public static void main(String args[])
   {
      College c = new College();
      Mobile m = new Mobile();
      System.out.println("College Messages:-");
      c.msg1();
      c.msg2();
      c.msg3();
      System.out.println("\nMobile Messages:-");
      m.alert1();
      m.alert2();
      m.recharge();
    }
}
```

Compile the program using the following command.

```
javac Student.java
```

Run the program using the following command.

```
java Student
```

Output: -

```
College Messages: -
Tomorrow is holiday due to Sunday
Last date to pay I Term fee is 10-10-2022!
Today we have a seminar at 3PM
```

Mobile Messages: -
your validity will be expired within two days
Recharge immediately for uninterrupted calls
Recharge with 75/- for 21 days unlimited calls

Explain exception handling in Java?

Definition: - An exception is an unexpected run time error that interrupts normal flow of execution and causes abnormal program termination. By using exception handling technique, we can deal with unwanted errors which are raised at runtime and allow the JVM to execute the remaining part of the program.
Generally, we have 2 types of errors in Programming. They are syntax errors and runtime errors. The syntax errors are modified at the time of compilation. But we should suffer with runtime errors. To solve this problem java provides exception handling technique. In this process, all executable statements were written in between "try" and "catch" keywords. The keyword "catch" is used to raise the exception when corresponding error occurs. Java supports two types of exceptions. They are

(i) Pre-Defined exceptions
(ii) (ii)User-Defined exceptions.

Predefined exceptions are classified into two types.
1) Asynchronous Exceptions: - These are used to deals with hardware problems and external problems. Some examples are …
1. Mouse Failure
2. Keyboard and Motherboard Failures
3. Memory Problems
4. Power Failures

Java.lang.Error is a *super class* of all asynchronous Exception Classes.

2. Synchronous exceptions: - These are used to deals with programmatic run time errors. **Java.lang.Throwable** is a super class of all synchronous Exception Classes. Synchronous Exceptions are divided into two types. They are …
1. Checked Exceptions
2. Unchecked Exceptions

1) Checked exceptions. Also called compile-time exceptions, the compiler checks these exceptions during the compilation process to confirm if the exception is being handled by the programmer. If not, then a compilation error displays on the system. Checked exceptions include SQLException, ClassNotFoundException, IOException, InstantiationException, InterruptedException, NoSuchMethodException, NoSuchFieldException, etc.

2) Unchecked exceptions. Also called runtime exceptions, these exceptions occur during program execution. These exceptions are not checked at compile time, so the programmer is responsible for handling these exceptions. Unchecked exceptions do not give compilation errors. Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, IllegalStateException, NullPointerException, NumberFormatException, StackOverflowError, ArithmeticException, etc.

/* Write a program using ArithmeticException */

```
class Eclass1
{
  public static void main(String  x[])
  {
   int  a=10,b=5,c=5,d;
   try
   {
      d=a/(b-c);
      System.out.println("Value of a/(b-c) = " + d);
   }
    catch(ArithmeticException e)
   {
      System.out.println("Dividing a number with zero is illegal");
   }
```

```
    }
}
```

/* Write a program using ArrayIndexOutOfBoundsException */

```
class Eclass2
{
   public  static  void  main(String  s[])
   {
      int  a[] = {12,23,34,45,56,90}
      try
      {
          int c = a[10]-5;
          System.out.println("Value = " + c);
      }
      catch(ArrayIndexOutOfBoundsException   e)
      {
           System.out.println("Array Overflow");
      }
   }
}
```

/* **USER DEFINED EXCEPTIONS**

We can create our own exception classes by extending the "Exception" class. The extended class contains attributes, constructors and methods like any other classs. The "throws" keyword is used while implementing a user-defined exception.

In the following example "IllegalValueException" is raised when student rno is less than zero or percentage of marks is greater than 100 */

```
import  java.io.*;
class IllegalValueException extends Exception
{
}
class Student
{
 int rno,pm;
 public Student(int  a, int  b)
 {
    rno=a;
    pm=b;
 }
 void show() throws IllegalValueException
 {
    if ( rno<=0  ||  pm>100 )
      throw new IllegalValueException();
    else
    {
     System.out.println("Roll Number = " + rno);
     System.out.println("Percentage of Marks = " + pm);
    }
  }
}
class Eclass3
{
  public  static void main(String  args[])
  {
    Student  s = new Student(12,181);
    try
    {
      s.show();
    }
    catch(IllegalValueException  e)
    {
     System.out.println("Invalid rno or percentage of
                                   marks found in student class");
    }
  }
}
```