Assignment - 1

1) Discuss the three object oriented programming principles.

→ The three OOP principles are :-

The OOP languages provide two mechanisms that help you implement object oriented model. They are encapsulation, inheritance and polymorphism.

Encapsulation : In java, encapsulation is a process of wrapping code and data transfer (together) into a single unit, for ex: A capsule that is mixed of several medicines.

we create a fully encapsulated class in java by making all the data members of the class private, Now we can use setter and getter methods to set and get the data on it.

when you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class, Specifically the data defined by the class are refferred to as member variables or instance variables.

Inheritance : In java, inheritance in java is a mechanism in which one object acquires all the properties and behaviours of parent object. The idea behind inheritance in java is that you can create new classes that are built upon existing class. when you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents a Is-A relationship, also known as parent child relationship.

Polymorphism: It is a concept by which we can perform a single action by different ways.

Polymorphism is derived from two greek words, 'poly' meaning many and "morphis" meaning form. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. we can perform polymorphism in java by method overloading and method overriding.

Assignment 2

2y Explain java application development stack and java virtual machine.

Ans :

→ A java application uses the java class libraries that are provided by the runtime environment to implement the application specific logic. The class libraries and, eventually in turn are implemented in terms of other class operations that are provided directly by the jvm. In addition, some applications must access native code directly.

Java virtual machines :

The jvm in the java run-time system and is the main component of making the java a platform independent language.

For building and running a java app. we need JDK (Java development kit) which comes bundled with java runtime environ-ment (JRE) and JVM. With the help of JDK the user compiles and runs his java program.

As the compilation of java program starts the java Bytecode is created i.e a class file is created by JRE. Bytecode is a highly optimised set of instructions designed to be executed by JVM.

Now the JVM comes into play which is made to read and execute this bytecode. The JVM is linked with operating System and reuses the bytecode to execute the code depending upon OS.

Therefore, a user can take this class file (Bytecode file) formed to any OS which is having a JVM installed and can run his programs easily without even touching the Syntax of a program and without actually having the Source code.

The class file, which consists of bytecode is not user understandable and can be interpreted by JVM only to built it into the machine code.

Assignment 3

3) Explain different access specifiers ?

→ Defaut : Whenever a specific access level is not specified, then it is assumed to be 'default'. The scope of the default level is within the package.

Public : This is the most common access level and whenever, the public access specifier is used with an entity, that particular entity is accessible throughout from within or outside the class, within or outside the package.

Protected : The protected access level has a scope that is within the package. A protected entity is also accessible outside the package through inherited class or child class.

Private : when an entity is private, then this entity cannot be accessed outside the class. A private entity can only be accessible from within the class.

Assignment 4

4) Differentiate the usage of access specifiers in Java and their scope.

→

Public Access Modifier	Private Access Modifier
→ This modifier is applicable for both top level classes and interfaces.	→ This modifier is not applicable for both top-level classes and interfaces.
→ Public members can be accessed from the child class of the same package.	→ Private members cannot be accessed from the child class of the same package.
→ Public members can be accessed from non-child classes of the same package, child class of outside package, non-child class of outside package.	→ Private members cannot be accessed from non-child classes of the same package, non-child classes of the same package, child class of outside package, non-child class of outside package.
→ Public modifier is the most accessible modifiers among all modifiers.	→ Private modifier is the most restricted modifier among all modifiers.

usage of access Specifiers in Scope.

fier Private Access Modifier

pli → This modifier is not applicable for both top-level classes and interfaces.

→ Private members cannot be accessed from the child class of the same package.

→ Private members cannot be accessed from non-child classes of the same package, non-child classes of the same package, child class of outside package, non-child class of outside package.

→ Private modifier is the most restricted modifier among all modifiers.

Protected Access Modifier

→ This modifier is not applicable for both top-level classes and interfaces.

→ Protected members can be accessed from the child class of the same package.

→ Protected members can be accessed from non-child of the same package, child class of the outside package, but we should use child reference only, and cannot be assessed from the non-child class of outside package.

→ Protected modifier is more accessible than the package and private modifier but less accessible than public modifier.

Assignment 5

5) Explain the process of compiling and running the java application with the help of hello world program.

→ Compiling the program

→ To compile the example program, execute the compiler, javac, specifing the name of the source file on the command line,

```
e:\> javac Example.java
```

The java compiler creates a file called example.class that contains the byte code version of the program.

The java bytecode is the intermediate representation of your program that contains instructions, The java virtual machine will execute these instructions

Thus the output of javac is not code that can be directly executed. To actually run the program, we must use the java app. launcher, called java.

To do so, pass the class name example as a command line argument as shown here,

```
C:\> java example
```

Then the output of the program is printed on the console.

Example:

```
c:\$ cat hello.java

class hello {
    public static void main (String args[]){
        System.out.println ("Hello, world!");
    }
}

c:\$ javac Demo.java
```

This command compiles the above program and generates a .class file.

```
c:\$ java Demo
```

This command executes the actual program.

O/p: Hello, world!

Assignment 06

6) Explain the scope and lifetime of a variable with an example.

→ As a general rule, variables declared inside a scope are not visible (i.e, accessible) to code that is defined outside that scope.

Thus, when you declare a variable within a scope you are localizing that variable and protecting it from unauthorised access and for modification. The scope rules provide the foundation for encapsulation. scopes can be nested, for example, each time you create a block of code, you are creating a new nested scope. when this occurs, the outer scope encloses the inner scope.

This means that objects declared in the outer scope will be visible to code within the inner scope. however, the reversed is not true.

Objects declared within the inner scope will not be visible outside of it.

To understant the effect of nested scopes, consider the following program.

Example :-

```
class scope {
  public static void main (String[] args) {
    int x; // known to all code within main
    x = 10;
    if (x == 10) {
      // start of new scope.
      int y = 20; // known only to this block
      // x and y both known here
      System.out.println ("x and y; " + x + " " + y);
      x = y * 2;
    }
    // y = 100; // through an error since y is
    // not known in this block

    // x is still know here
    System.out.println ("x is " + x);
  }
}
```

Assignment 7

7) Explain how array in java work differently than c or c++. write a java program to display a given matrix

0 1 2 3 5
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19

→ In java, Arrays can be of any type and can also be multi-dimensional. An element in an array is accessed by its index. Arrays offer a convinient means of grouping related information.

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and array_var is the array variable that is linked to the array.

```
array_var = new type [size];
ex:   arr = new int [10];
```

Keyword new is used to allocated memory for the array, you must specify the type and no. of elements to allocate

Note : All elements in the array will be initialized to zero.

Program for displaying the matrix

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

int[][] arr = {{0,1,2,3,4},{5,6,7,8,9},
               {10,11,12,13,14},{15,16,17,18,19}};
for(int i=0; i<4 ; i++){
   for(int j=0; j<5 ; j++){
      System.out.println(arr[i][j]);
   }
}
}
}
```

Assignment 8

8) How arrays are defined and initialized in Java explain with an example.

→ One dimentional array is, essentially a list of a like type variables.

To create an array you first must create an array variable of the desired type. The general form of a one dimensional array declaration is

type var-name[];

eg: int month_day[];

Although this declaration establishes the fact that month-days is an array variable, no array actually exist.

In fact, the value of month-days is set to null, which represents an array with no value.

To link month_days with an actual, physical array of integers, you must allocate memory using "new" and assign it to month_days.

"new" is a special operator that allocates memory

array-var = new type[size];

Here, 'type' specifies the type of data being allocated, 'size' specifies the number of elements in the array, and array_var is the array variable that is linked to the array.

To use 'new' you must specify the type and size to allocate.

month_days = new int[12];

now 'month_days' will refer to an array of 12 integer.

An array can be initialised in two ways

```
int arr[] = { 1, 2, 3, 4 }
```

Here an array arr is both declared and initialised at once.

```
int arr[] = new int [4];
arr[0] = 1; arr[1] = 2; arr[2] = 3; arr[3] = 4;
```

Here first the array is initialised and then each element is initialised one by one using array index.

An array can be initialised in two ways

```
int arr[] = {1, 2, 3, 4};
```

Here an array arr is both declared and initialised at once.

```
int arr[] = new int [4];
arr[0] = 1; arr[1] = 2; arr[2] = 3; arr[3] = 4;
```

Here first the array is initialised and then each element is initialised one by one using array index.

Assignment 09

Q) What is type casting? Explain different types of type casting in java.

→ In type conversion, if the two types are compatible, the java will perform the conversion automatically.

For example, it is always possible to assign an int value to a long variable.

However not all types are compatible, and thus, not all type conversions are implicitly allowed.

To create a conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion. It has this general form

(target_type) value

Here target_type specifies the desired type to convert the specified value to

For example :→ The following fragment casts int to a byte, If integers value is larger than the range of a byte. it will be reduced modulo (the remainder of an integer division by the byte's range).

```
int a;
byte b;
b = (byte) a;
```

```
int a = 30;
long b;
b = a;
```

When a floating point value is assigned to an integer type, the fractional component is lost

```
int a = 30.5
System.out.println(a);
O/P = 30
```

Types of type casting :

→ Widening type casting (Implicit) :
Converting a lower data type into a higher one is called widening type casting. It is also known as implicit conversion or casting down. It is done automatically. It is safe because there is no chance to lose data.

→ Narrowing type casting (Explicit) :
Converting a higher data type into a lower one is called narrowing type casting. It is also known as explicit conversion or castingup. It is done manually by the programmer. If we do not perform casting then the compiler reports a error.

Assignment 10.

10) List out the four main differences between procedure oriented and object oriented programming.

Ans

Procedure Oriented programming	Object oriented programming
→ In pop, importance is not given to data but to functions as well as sequence of actions to be done	→ In oop, importance is given to the data rather than procedures or functions because it works as a real world.
→ In pop, program is divided into small parts called functions	→ In oop, the program is divided into parts called objects.
→ In pop, most functions uses global data for sharing that can be accessed freely from functions in the system.	→ In oop, data can not move easily from function, it can be kept public or private so we can control the access of data.
→ In pop, data can move freely from function in the system.	→ In oop, objects can move and communicate with each other through member functions.
→ pop does not have any proper way for hiding data so it is less secure.	→ oop provides data hiding so provides more security.

Assignment 11

11) Explain type casting. Is java Strongly typed language? justify your answer.

→ → Java is a strongly typed language.
→ Indeed, part of java's safety and robustness comes from this fact.
→ First every variable has a type, every expression has a type, and every type is strictly defined.

→ Second, all assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.

→ The java compiler checks all expression and parameters to ensure that the types are compatible.

→ Any type mis matches are errors that must be corrected before the compiler will finish compiling the class.

Assignment 02

12) Mention the different integer types supported by java with their memory requirements. Explain how decimal, Octal, hexadecimal literral can be initialized.

→

Integer type	Size	Description
Byte	1 byte	Stores whole numbers from -128 to 127
Short	2 bytes	Stores whole numbers from -32,768 to 32,767
Int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647.
long	8 bytes	Stores whole numbers from -9×10^{18} to 9×10^{18}

A hexadecimal number is a value in base-16 There are 16 digits, 0-9 and the letters A-F (case does not matter)
A to F represents 10 to 15

```
int hEX = 0x5E;
```

'0x' prefix is used for hexadecimals.

A binary number is a value in base 2 and uses the digits 0 and 1.

int BIN = 0b1101110

'0b' prefix is used for binary literals.

A Octal number is a value in base 8 and uses the digits 0-7

int OCT = 0156;

'0' (alphabet '0') is used as prefix for octal numbers.

Assignment 13

13) Explain different type of arrays with declaration.

→ There are two types of arrays :
→ One dimensional
→ Multi-dimensional

→ One-D arrays
It is essentially a list of like type variables. To create an array you first must create an array variable of the desired type.
The general form of a one-D array declaration is
type var-name[];
eg:- int month_days[];

→ Multi-D arrays:
In Java, multidimensional arrays are actually arrays of arrays.
To declare a multidimensional array variable, specify each additional index using another set of square brackets.
for example: The following declares a two dimensional array variable called twoD.

```
int twoD[][] = new int[4][5];
```

Creates four arrays with each array can store upto five elements.

Q) Explain the overloading the methods with example.

Ans: Having more than one method with a same name is called as method overloading.

→ To implement this concept, the constraints are: the number of arguments should be different, and/or type of the arguments must be different.

→ Note that, only the return type of the method is not sufficient for overloading.

```
class overload
  {
    void test()
      {
        System.out.println("no parameters");
      }
    void test(int a)
      {
        System.out.println("Integer a:" +a);
      }
    void test(int a, int b)
      {
        System.out.println("With two arguments " +a+ " " +b);
      }
    void test(double a)
      {
        System.out.println("double a:" +a);
      }
  }
Public class OverloadDemo
  {
    public static void main(String args[])
      {
        Overload ob = new overload();
```

```
        ob.test();
        ob.test(10);
        ob.test(10, 20);
        ob.test(123.25);
    }
}
```

Output:-

```
No Parameters
Integer a: 10
with two arguments: 10 20
double a: 123.25
```

2) Explain with an example how to pass objects as parameters.

Ans: Just similar to primitive type, even object of a class can also be passed as a parameter to any method.

→ The "this" keyword refers to the current object in a method or constructor. The most common use of this keyword is to eliminate the confusion between class attributes and parameters with the same name.

```
class Test
  {
    int a, b;
    Test(int i, int j)
      {
        a = i;
        b = j;
      }
    boolean equals(Test ob)
      {
        if(ob.a == this.a && ob.b == this.b)
            return true;
        else
            return false;
      }
  }
  class PassOb
    {
      public static void main(String args[]){
            Test ob1 = new Test(100, 22);
            Test ob2 = new Test(100, 22);
            Test ob3 = new Test(-1, -1);
      System.out.println("ob1 == ob2:" + ob1.equals(ob2));
      System.out.println("ob1 == ob3:" + ob1.equals(ob3));
        }
  }
```

Output: ob1 == ob2: true
ob1 == ob3: false

3) Explain Recursion with an example.

Ans: A method which invokes (calls) itself either directly or indirectly is called as recursion method.

* Every recursion method should satisfy the following condition:

→ It should have atleast one non-recursive terminating condition.

→ In every step, it should be nearer to the solution (that is problem size must be decreasing).

```
class Factorial
  {
     int fact(int n)
       {
          if(n==0)
           return 1;
        else
          return n*fact(n-1);
       }
  }
  class FactDemo {
     public static void main(String args[])
       {
          Factorial f = new Factorial();
  System.out.println(" Factorial of 3 is: " +f.fact(3));
  System.out.println(" Factorial of 8 is: " +f.fact(8));
    }
}
```

Output:- Factorial of 3 is: 6
Factorial of 8 is: 40320

4) Explain Exception handling with example.

Ans: The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Exception :- It is an abnormal condition, also it is an event that disrupts the normal flow of the programs. It is an object which is thrown at runtime.

Exception Handling :-

It is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.,

The core advantage of exception handling is to maintain the normal flow of the application.

→ Java provides five keywords that are used to handle the exception.

i) try :- The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

ii) catch :- The "Catch" block is used to handle the exception. It must be proceded by try block which means we can't use catch block alone. It can be followed by finally block later.

iii) finally :- The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

iv) throw :- The "throw" keyword is used to throw an exception.

7) Throws:- The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Ex:
```
class Demo{
    Public static void main(String args[])
    {
    int[] marks={97,98,99};
      try
      {
      System.out.println(marks[5]);
      }
      catch(Exception exception)
      {

      }
    System.out.println("Exception Handled");
    }
  }
}
```

Output:- Exception Handled.

5) Explain call by value and call by reference with example.

Ans: Call by value:-

Call by value means calling a method with a parameter as value. Through this, the argument value is passed to the parameter.

In this, the modification done to the parameter passed does not reflect in the caller's scope. While in the The values of the arguments remain the same even after the method invocation.

Eg:

```
class Operation{
    int data = 50;

void change(int data){
 data = data + 100; // Changes will be in local variable only
 }

public static void main(String args[])
 {
     Operation op = new Operation();

System.out.println("before change" + op.data);
    op.change(500);

System.out.println("after change" + op.data);
 }
}
```

Output:

```
before change 50
after change 50
```

Java uses only call by value while passing reference variables as well. It creates a copy of references and passes them as values to the methods.

* Call by reference:-

Call by Reference means calling a method with a parameter as a reference. Through this, the argument reference is passed to the parameter.

The modification done to the parameter passed are persistent and changes are reflected in the caller's scope.

As reference points to the same address of object, creating a copy of reference is of no harm. But if new object is assigned to reference it will not be reflected.

Ex:

```
class Operation2{
    int data=50;

  void change(Operation2 op){
    op.data=op.data+100;    // changes will be in the
                            // instance variable.
  }

Public static void main(String args[])
        {
    Operation2 op=new Operation2();

System.out.println("before change"+op.data);
    op.data(500);

System.out.println("after change"+op.data);
  }
}
```

Q: Explain Access specifiers in Java.

Ans: Java provides entities called "Access specifiers" that help us to restrict the scope or visibility of a package, class, constructor, methods, variables or other data members.

By using the access specifiers, a particular class method or variable can be restricted to access or hidden from the other classes. The access specifiers also determine which data members (methods or fields) of a class can be accessed by other data members of classes or packages etc.

→ Java provides four types of access specifiers are:

a) Default b) Public c) Protected d) Private

a) Default: whenever a specific access level is not specified, then it is assumed to be "default". The scope of the default level is within the package.

```
class BaseClass
{
  void display()
  {
    System.out.println("Baseclass::Display with 'default'
                        scope");
  }
}
  public static void main(String args[])
  {
    BaseClass obj = new BaseClass();
    obj.display();
  }
}
```

Output:- Baseclass::Display with 'default' scope

b) Public :- This is the most common access level and whenever the public access specifier is used with an entity, that particular entity is accessible throughout javaw within or outside the class, within or outside the package etc.

```
class A
{
  public void display()
  {
    System.out.println("Software Testing!");
  }
}
class main
{
  public static void main(String args[])
  {
      A obj = new A();
      obj.display();
  }
```

Output :- Software Testing!

c) Protected :-

The protected access level has a scope that is within the package. A protected entity is also accessible outside the package through inherited class or child class.

It doesn't matter whether the class is in the same package or different package, but as long as the class that is trying to access a protected entity is a subclass of this class, the entity is accessible.

Note that a class and an interface cannot be protected i.e., we cannot apply protected modifiers to classes and interfaces.

Ex:
```
class A
{
    protected void display()
    {
        System.out.println("This is Protected");
    }
}

class B extends A{}
class C extends B{}

class Main{
    public static void main(String args[])
    {
        B obj = new B();
        obj.display();
        C cobj = new C();
        cobj.display();
    }
}
```

Output:- This is Protected
This is Protected.

d) Private:-

When an entity is private, then this entity cannot be accessed outside the class. A private entity can only be accessible from within the class.

→ Private access specifiers cannot be used for classes and interfaces.

→ The scope of private entities (methods + variables) is limited to the class in which they are declared.

→ A class with a private constructor cannot create an object of the class from any other place like the main method.

```
Test class Test
    {
      int a;
      public int b;
      private int c;
    void setc(int i)
      {
        c=i;
      }
    int getc()
      {
        return c;
      }
    }

class AccessTest
    {
      public static void main(String args[])
        {
          Test ob = new Test();
            ob.a = 10;
            ob.b = 20;
      // ob.c = 100;    // inclusion of this line is Error!
          ob.setc(100);
    System.out.println("a, b and c: " + ob.a + " " + ob.b + " "
                                    + ob.getc());
        }
}
```

Output:- a, b and c: 10 20 100

7) Explain static method in Java:-

Ans: When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.

→ Instance variables declared as static are global variables.
→ When objects of this class are declared, no copy of a static variable is made.
→ Instead, all instances of the class share the same static variable.

* Methods declared as static have several restrictions:

→ They can only call other static methods.
→ They must only access static data.
→ They cannot refer to this or super in any way.
→ If you need to do computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded.

Ex:

```
class UseStatic
{
    static int a = 3;
    static int b;

    static void meth(int x)        // static method
    {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }
    static
    {
        System.out.println("static block initialized");
        b = a * 4;
    }
```

```
public static void main(String args[])
    {
        meth(42);
    }
}
```

Output:- static block initialized
x = 42
a = 3
b = 12

→ Outside of the class in which they are defined, static methods and variables can be used independently of any object.

→ To do so, you need only specify the name of the class followed by the dot operation.

→ The general form is: classname.method();

```
class StaticDemo
    {
        static int a = 42;
        static int b = 99;
    static void callme()
        {
        System.out.println("Inside static method, a = " + a);
    }
}
```

Output:- Inside static method, a = 42
Inside main, b = 99

Assignment -25

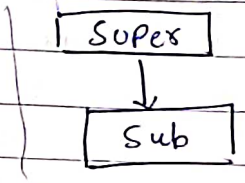
-) Explain types of inheritance with example

Ans: When we create a new class from the existing class such that the new class access the features and properties of existing class called inheritance

Types of inheritance:

-) Single
-) Multilevel.

-) Single - Single inheritance which contain only one super class and only one sub class.

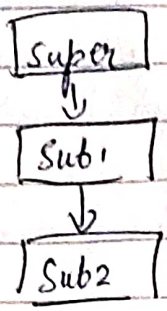


Ex:

```
class student
int roll, mark;
 String name;
void input()
{ SOP ("Enter roll name & mark:"); }}
class ankit extend student
{
   void disp()
   {
      roll = 69;   name: Ankit ;  marks = 69;
   SOP ( roll+ " " + name + " " + marks);
}
  Public static void main (String[] args) {
      ankit x = new ankit();
      x. input();   x. disp();
```

o/p = Enter roll & marks
69 Ankit 69

Multilevel inheritance :- Which is having only one super class and multiple sub classes




```
Ex.   class A
        int a, b, c;
Void add() { c = a+b;
{       SOP ("Sum of Two number = "+c)
    a = 10; b = 20;
    c = a+b;
      SOP (" Sum of Two number: "+c); } }
class B extends A
  {
    void Sub ()
    {
      a = 200 ; b = 100;
       c = a-b;
      SOP (" sub of two number; "+d); } }
class C extend B
  {
    void multi ()
    {
      a = 10 ;  b = 20;
       c = a+b;
      SOP (" Multiplication of two number : +c);
    }
  }
```

o/p =

```
Sum of two number : 30
Sub of two number : 100
Multiplicat of two number : 200
```

```
class Test {
    Public static void main (String [] args){
        C r = new C();
        r.add(); r.sub(); r.multi();
```

Assignment - 26

→ Explain Overriden with an example.

Ans Whenever we writing a method in super class and subclass in such a way that method name and parameters must be same called method overriding

Syntax :-

```
class A
{
    void show()
    {
    }
}
class B extend A
{
    void show()
    {
    }
}
```

Ex:

```
class shape
{
    void draw()
    { SOP ("can say shape type") }}
class squre extends shape {
    @override
    void draw()
    { SOP (" Square shape") }}
class Demo
{
    Public static void main (String [] args){
        shape s = new square();
        s. void draw ();  }
```

O/P :- square shape

Assignment 27

-> Explain abstract class with an example

A class which contains the abstract keyword in its declaration is called abstract class

-> we can't create the object of abstract class

-> It may or may not contain abstract method

-> It can have abstract & non-abstract method

-> To use an abstract class, you have to inherit it from sub classes

Ex:- ~~abstclass animal~~

Ex:
```
abstract class animal
{
}
class Demo
{
    Public Static void main (String args[])
    {
        animal a = new animal ();
    }
}
```

o/p = 1 error

∴ we can't create object of abstract class

-> Since we can create reference

Ex:
```
abstract class animal
{
}
class Dog extends animal
{
}
class Demo
{
    Public static void main (String args[])
    {
        animal a=new Dog ();
    }
}
```

o/p : no error. it compiles successfully.

Assignment - 28

4) Explain the packages. explain importing the packages?

-) Package: A package is collection number of classes interfaces and sub package of same type into a particular groups.
-> Package is nothing but folder in windows

Types
-> User - defined
-> Pre - defined

1) User defined - The package which is created by Java programmers or user for their own use.
Syntax : package package-name;
Ex:

2) Pre defined - The package which are created by java developer people are called pre-defined package.

Ex: java.util, java.lang

Importing Packages.

-> We have to write import keyword to import packages (name of the package)
Ex: import. java.util.*;
-> This will import all the classes from the util package
-> To import specific class from inside the util package we have to write the name of the class
Ex- import java.util.scanner;
-> Similarly we can import users defined packages

Ex :

```
package MyPackage;
class Test
{
  int a, b;
  Test (int x, int y)
  {
    a = x ; b = y;
  }
  void disp()
  {
    SOP ("a= " + a + " b= " + b);
  }
}
class PackDemo
{
  public static void main (String args[])
  {
    Test t = new Test(2,3);
    t.disp();
  }
}
```

o/p = a=2 b=3

Assignment 29.

5) What is interface explain with example?

Ans Interface is just like a class which contains only abstract method or collection of abstract method is called interface

-> To acheive interface Java provides keyword called implements.

-> Interface methods are by default public & abstract

-> Interface variable are by defaut public + static + final

-> Interface method must be overriddden inside the implementing classes

-> Interface nty but deals btw client & developer.

Eg:-

```
interface Abc
{
    void show();
}
class AbcImpl implements Abc
{
    Public void show()
    {
        SOP ("Hello world");
    }
}
Public class Demo
{
    public static void main (String args[])
    {
        Abc obj = new AbcImpl();
        obj.show();
    }
}
```

Ofp :- Hello world.

Assignment 30

6. Difference between interface & abstract class.

Interface	Abstract
-) It contains only abstract method	-) It contain both abstract & non-abstract method.
-) It supports multiple inheritance	-) It does not support multiple inheritance
-) By default interface are public + abstract	-) There are no restriction on abstract class method
-) By default interface variable are public + static + final	-) No need to declare variable as public + static + final.
-) Interface keyword is used to declare interface	-) Abstract keyword is used to declare abstract class
-) it doesn't allow to declare constructor	-) abstract class can declare constructor

Assignment - 31

→ Explain five keywords of exception handling

Exception : is unexpected situation that occured at runtime

Exception handling : is method in which we should have alternate source through which we can handle Exception

5 Keywords of exception handling

1) Try
2) catch
3) finally
4) throw
5) Throws.

1) Try → The 'try' keyword is used to specify the block where we should place an exception code
→ we cannot use try block alone it must be followed by either catch or finally.

2) catch → The 'catch' block is used to handle the exception. it must be preceded by try block which means we can't use catch block alone it must be followed by final block later.

3) finally → The finally block is used to execute the necessary code of the program. it is executed whether an exception is handled or not,

4) Throw - The throw keyword is used to throw an exception

5) Throws - The throws keyword is used to declare exceptions. It specifies that there may occur an exception in the method.

Assignment - 32

=> Explain throw and throws keyword:

=> Throw :- ① Throw keyword is used to throw an exception

② Throw always present inside the body

③ Ex :- void m1()
{
throw new AE();
}

④ We can throw one exception at a time

⑤ Throw is followed by instance

=> Throws :- ① Throws keyword is used to declare an exception as well as by pass the caller

② Throws keyword always used with method signature

③ Ex :- void m1() throws AE
{

}

④ It can handle multiple exception using throws keyword

⑤ Throws is followed by class.