ARFA REHMAN

JAVA NOTES

* Characteristics of Java

1. Compiled and Interpreted :

Usually a computer language is either interpreted or compiled. Java combines both the approches and makes itself a two staged system. First, the java compiler convert the source code into bytecode instructions. As bytecodes are not machine codes, therefore in the second stage, the java interpreter generates machine code that can be executed by the machine that is running a java program. Thus java is both a compiled and an interpreted language.

2. Machine Independent & Portable :

The most significant contribution of Java over other languages is its portablity. A java program can easily be moved from one computer to another anytime and anywhere. Changes or upgrades in operating system, processor and system resources will not force any changes in the java program. That is why java is the most popular language for programming on the Internet which interconnects different kinds of systems worldwide. We can download a java applet from any remote computer onto our local system via internet and execute it locally. This made Internet an extension of User's basic system providing practically unlimited number of accessible language applets and applications.

Java provide portablity in two ways. First, java compiler generate bytecode that can be implemented on any

machine. Second the size of primitive data types are machine independent.

3. Object Oriented:

Java is a true object oriented language. Almost everything in java is an object. All program codes and data resides within classes and object. Java comes with an extensive set of classes, are arranged in packages that can be used by programs through inheritance. The object model of Java is simple and easy to extend.

4. Robust and Secure:

Java is a robust language. It provide many safeguards to ensure a reliable code. It has strict runtime and compile time checking for data type. It is designed as a garbage collected language relieving the programmers virtually all memory management problems. It also incorporates the concept of exception handling that captures the series of errors and eliminates any risk of crashing the system.
Security is a big issue for any programming language that is used on internet. Threats of viruses and abuse of resources are everywhere. Java System not only verify memory access but also ensures no viruses are communicated with any applet. The absence of pointers in Java ensures that programs can not gain access to memory location without proper authorization.

5. Distributed:

Java is designed as a distributed language for creating application on networks. It has the ability to share both data and program. Java applications can open and access any remote object on Internet as easily as they do in any local system. It enables multiple users at multiple loca remote locations to collaborate and work together on a single project.

6. Small, Simple and familiar:

Java is a small and simple language. Many features of C and C++ that are either redundant or source of unreliable code are not part of Java. For example - java does not supports pointers, prepoc processors header files, goto statement etc. It also eliminates operator overloading and multiple inheritance.
Familiarity is another striking feature of java. To make it look familiar language to the existing programmers, it was modelled on C and C++. Java uses many constructs of C and C++ therefore a java code looks like a C++ code. Infact, it is a simplified version of C++.

7. Multi threaded and interactive:

Multi threaded means handling of multiple task simultaneously. Java programs are multi threaded. That means we need not to wait for a job's app. to finish its task before begining another. For ex- we can listen to an audio clip while scrolling

in page and at the same time downloading an applet from a distant computer. This features greatly improves the interactive performance of a graphical application.
Java runtime provides tools that supports multi process synchronization and smoothly running interactive systems.

8. High Performance.

Java performance is impressive for any interpreted language, mainly due to the use of intermediate bytecodes. According to Sun, Java speed is comparable with to its the native C and C++. Java architecture is designed to reduce overheads during runtime. Further the incorporation of multireading enhances overall execution speed of Java program.

11 12

9. Dynamic and Extensible:

Java is a dynamic language. It is capable of dynamically ~~in~~ linking ~~&~~ in new class libraries methods and objects. It can also determine the type of class through query, making it possible to either dynamically link or abort the program, depending upon response.
Java supports the functions written in other languages such as C, C++. These are known as native methods. It enables a programmer to use the efficient functions available in these languages. Native methods are dynamically linked at runtime.

13

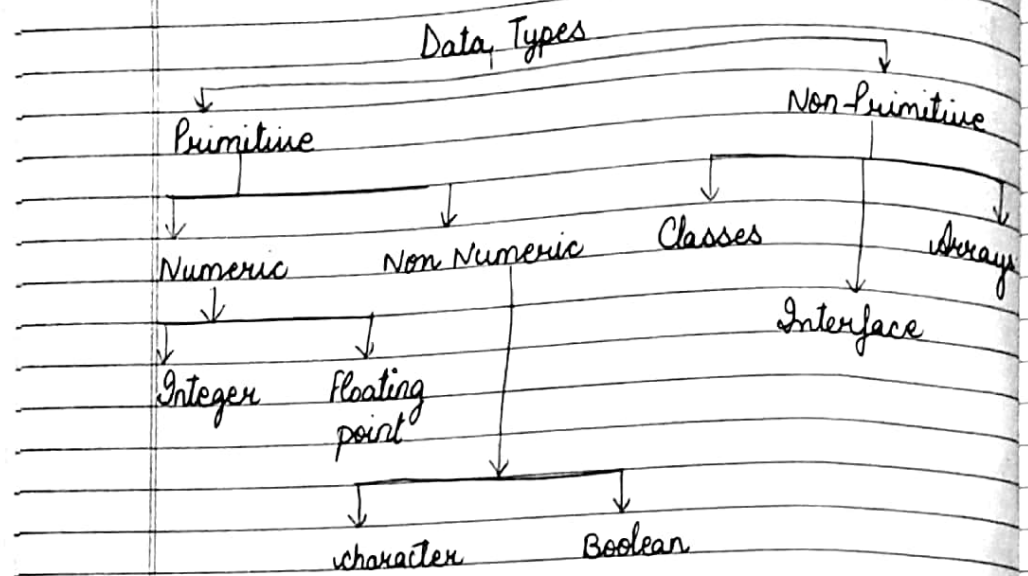
10. Ease of Development:

Java 2 Edsl Standard Edition. supports features such as Generics, Enhanced for loops, Autoboxing or unboxing, Typesafe Enums, Varargs, static imports and Annotations. This reduces the work of a programmer by shifting the responsibility of reusable code to the compiler. The resulting source code is free from bugs b'coz errors made by compiler are less when compared to that of programmer. Thus, each linguistic feature is designed to develop a java program in an easier way.

Data Types.

Every variable in Java has a data type. Data type specify the size and the type of value that can be stored. Java is rich in data types which are explained below.

1. Primitive / Intrinsic or built-in data types
   i. Numeric - a) Integer
                b) floating point
   ii. Non Numeric - a) character
                     Boolean
2. Non-Primitive / Derived - 1. Classes
                             2. Interface
                             3. Arrays.



1. Primitive Data types:

Primitive data types refers to built in or intrinsic data types. They can be grouped into four categories -

1. Integers
2. floating point numbers
3. Characters
4. Boolean

i. Integers:

Java defines four integer Types - byte, short, int, long. All these are signed, positive and negative values. Java does not support unsigned, positive only integers. The width and size of these integer types varies widely.

i. Byte - This is smallest integer type. It is signed 8-bit type that has range from -128 to 127. These are useful when one working with a stream of data from a network of file

Declaration	Example
byte var1, var2;	byte a, b;

ii Short - short is signed 16-bit type. It has range from -32,768 to 32,767. It is probably least used data type.

Declaration	Example
short var1, var2;	short a, b;

iii int - Most commonly used integer type. It is 32 signed 32 bit type that range from -2,147,483,648 to 2,147,483,647.

iv. long - long is signed 64-bit type and used where int type not large enough to hold a value. Its range is quite large. This makes it useful when large whole numbers are required.

ii. floating point Value:

floating point number also known as real numbers are used when evaluating expressions that require fractional part. There width and range varies

1. float - this specifies single precision value that uses 32 bits of storage useful when fraction component is needed but not the precision.

2 double - specifies double precision and required 64 bit for storage.

3. Characters :

In java data types to store characters is char. In java char is 16-bit type and it uses unicode to represents characters that defines international characters too to make java globally portable.

4. Boolean :

Java has primitive data type, boolean, for logical data type values. It can have one of two possible values either true or false required by relational operators and conditional statement.

ii Non primitive Data types

These are not defined by the Programming language but are instead created by the programmer also called as reference data types.

1. Classes - A class is a blue print from which individual objects are created. It can have no. of functions methods to access the value of various kinds of methods

2. Array - Collection of homogeneous or hetrogeneous Type of data.

3. Interfaces : It is a collection of abstract methods. It is somewhere similar to a class but the difference is it only contains constants, default and static methods.

8. Operators :

Operators are symbols that tells the compiler to perform certain mathematical or logical manipulations. They usually a part of any expression or logical or mathematical.
Java operator can be classified into following

1 Arithmetic Operator
2 Relational Operator
3 Logical Operator
4 Assignment Operator
5 Increment / Decrement Operator
6 Conditional Operator
7 Bitwise Operator
8 Special Operator

1. Arithmetic Operators

These are used to construct mathematical expression as in algebra. Java provide all basic arithmetic operators

+ Addition & Unary plus
- Subtraction & Unary minus
* Multiplication
/ Division
% modulus (Remainder)

2. Relational Operators

These are used to compare to quantities depending on their relation. Java support six relational operators.

<	less than
>	greater than
<=	less than equal to
>=	greater than equal to.
==	equal to eg
!=	not equal to.

3. Logical Operators

These are used to form compound conditions by combining two or more relations

&&	logical AND
||	logical OR
!	logical NOT.

4. Assignment operators :

These are used to assign the value of an expression to a variable. we generaly use '=' for assignment but in java 'op' is also used as shorthand assignment Operator.

```
v op= exp;
```

Where 'v' is a variable 'op' is binary operator.

5. Increment / Decrement Operator:

These are the operators which are used to increase or decrease the value of a variable by '1' They are unary operators.

6. Conditional Operator :

The character pair ?: is a ternary operator. It is used to construct conditional expressions of the form.

```
exp1 ? exp2 : exp3.
```

7. Bitwise Operators

These are used to manipulate a data at values of bit level Bits are tested either by shifting left or right.

&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right
>>>	shift right with zero fill.

8. Special Operators :

Java supports two special operator such as instanceof and member selection operator.

1. Instance Operator returns true if the object on the left side is an instance of class given in left side.

2. Dot operator is used to access the instance variable and method of class object

J.V.M ~~and~~ ~~JDK~~

J.V.M refers to Java Virtual Machine is an abstract computing machine ~~that enables a computer that~~ enables a computer to run a java program. There are three notions ~~to run a java pro~~ of J.V.M. specification, implementation and instance. The specification is a document that formally describes what is required ~~meet~~ of a J.V.M implementation. A J.V.M implementation is a computer program that meets the requirements of J.V.M specification. An instance of JVM is an implementation running in a process that executes a computer program compiled into Java bytecode.

JDK:

Java development Toolkit is ~~an~~ a collection of tools that are used for developing and running java programs. They include

1. appletviewer for viewing java applets
2. javac is java compiler
3. java is java interpreter
4. javap is java disassembler
5. javah for C header file
6. javadoc for creating HTML documents
7. jdb Java debugger

To create a java program, we need to create a source code file using a text editor. The source code is then compiled using the java compiler javac and executed using the java interpreter java. The java debugger, jdb is used to find errors. A java compiled program can be converted into source code with the help of java disassembler javap

Java Runtime Environment

The java Runtime environment facilitates the execution of programs developed in java. It primarily comprises of the following:

1. J.V.M - It is a program that interprets the immediate java bytecodes and generates the desired output. It is because of byte code and JVM concepts that programs written in java are highly portable.

2. Runtime Class libraries: These are a set of core class libraries that are required for the execution of Java programs

3. User Interface toolkit: AWT and Swing are examples of toolkits that supports varied input methods for the users to interact with the application program.

4. ~~Java plug-in~~ Deployment technologies - JRE comprises the following key deployment technologies:
   1. Java plug-in - Enables the execution of java applet on ~~a~~ browse
   2. Java Web Start - Enables remote deployment of an app

* Objects and classes:

Basic component of a Java program

Objects: An entity that has state and behaviour is known as object. It is an instance of a class. An object has three characteristics:

1. state: represents data (value) of an object.
2. behavior: represents the functionality of an object.
3. identity: It is a unique ID used by J.V.M to identify each object uniquely.

creation and declaration of an object:

1. Declaration of an object is done as:

```
classname   Object_name;
Ex  Time    t1;
```

2. Creation of an object is done as;

```
Object_name = new class_name();
Ex    t1    = new Time();
```

The 'new' keyword is used to create an object.

3. Alternatively we can combine declaration and creation in one statement as follows:

```
class_name Object_name = new class_name();
Ex  Time   t1          = new Time();
```

Accessing class members outside the class

```
Obj_name.Variable name = Value
t1.t = 10;
obj_name.method_name(parameter_list);
t1.total(5,60,20);
```

class: A class is a template or blue print from which objects are created. It provides a convinient way for packaging together a group of logically related data items and functions that work on them. Classes create objects and objects use methods to communicate between them.

In Java, the data items are called 'fields' and the functions are called methods. Calling a specific method in an object is described as sending the object a message.

syntax –

```
class <class_name>
{
  field;    // declaration & definition
  method;
}
```

Example –

```
class Rectangle
{
  int len, wid;                    // Declaration of Variable
  void getdata(int x, int y)       // method declaration
  {
    len = x;
    wid = y;                       // method definition
  }
  void area()
  {
    int area = len * wid;
  }
  void disp()
  {
```

```
System.out.println("Length = " + len);
System.out.println("Breadth = " + wid);
System.out.println("Area = " + area);
}
}

Class rect
{
  public static void main (String args[])
    {
      Rectangle r1 = new Rectangle ();
      Rectangle r2 = new Rectangle ();
      r1.getdata (2,4);
      r1.area ();
      r1.disp ();
    }
}
```

Accessing class methods / Accessing class fields

Output -

```
Length = 2
Breadth = 4
Area = 8
```

Inheritance:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviour of another object (parent object). Or we can say it is a technique of deriving a new class from an old class.
It represents IS A relationship or 'Parent-Child' relationship.

The old class is known as 'Super class' and the new one is known as 'Sub class'.

We use inheritance in Java for reusing the properties of existing class, for method Overriding, so that the runtime polymorphism can be achieved.

Syntax:

```
class subclass extends superclass-name
  {
    // methods and field
  }
```

The 'extends' keyword indicates that you are making a new class that derives from an existing class. The meaning of 'extends' is to increase the functionality

Example:

```
class Emp
  {
    float salary = 40000;
  }
class Emp1 extends Emp
  {
    int bonus = 10000;
    public static void main (String args[])
      {
        Emp1 e1 = new Emp1 ();
        System.out.println ("Salary of employee = " +
                            e.salary);
        System.out.println ("Bonus received = " + e.
                            salary);
```

}
}

Salary of employee = 40000.0.
Bonus received = 10000.

In the above example, Object of Emp 1 'c' can access the field of own class as well as emp class.

Types of Inheritance

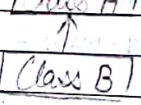
There are following types of inheritance supported by Java.

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

Multiple Inheritance and Hybrid inheritance are achevied by the means of 'Interface'.

1. Single Inheritance

When a class extends another class (only one class) then it is called as Single inheritance.



Class B extends only one class A. Here class B will be the Subclass and class A will be Super class.

Example:

```
class Room
  {
    int l;
    int b;
    Room (int x, int y)
      {
        l = x;
        b = y;
      }
    int area()
      {
        return (l*b);
      }
  }
class Bedroom extends Room // Inheriting room
  {
    int h;
    Bedroom (int x, int y, int z)
      {
        super (x,y);   // pass value to super class.
        h = z;
      }
    int volume()
      {
        return (l* b* h);
      }
}
class inheritest
 {
   public static void main (String args[]).
```

```
                    room1
Bedroom b = new Bedroom (14, 18, 20);
int area1 = room1.area();   // Super class method
int volume1 = room1.volume(); // base class method
System.out.println ("Area1 = " + area1);
System.out.println ("Volume = " + volume);
}
}
```

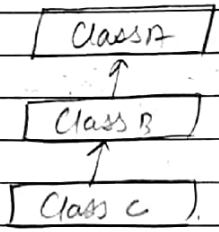
Output-

Area = 168

Volume = 1680

2. Multi-level Inheritance

In multi-level inheritance a derived class will be inheriting a parent class and as well as the derived class act as the parent class to other class.



Class B inherits the property of class eg A and again class B act as parent for class C. In short class A parent for class B and class B parent for class C.

Program.

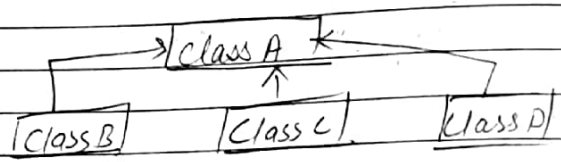
```
public class A
  {
    public void dispA()
      {
        S.o.p ("disp() A");
      }
  }
public class B class-B extends class_A
  {
    public void dispB()
      {
        S.o.p ("disp() B");
      }
  }
public class classC extends class B.
  {
    public void dispC()
      {
        S.o.p ("disp() C");
      }
  }
  Public static void main (String args[])
    {   // assigning class object to class reference
      classC c = new ClassC();
      c.dispA(); // c object calling method of
      c.dispB(); //   parent classes -
      c.dispC(); //
    }
}
```

Output:

dispA
dispB
dispC

Hierarchical Inheritance

In this one parent class will be inherited by many sub classes.



Class A will be inherited by class B, class C and Class D.

Example:

```
public class ClassA
    {
        public void dispA()
            {
                S.o.p ("dispA");
            }
    }
public class ClassB extends ClassA
    {
        public void dispB()
            {
                S.o.p("dispB");
            }
```

```
public class ClassC extends ClassA
    {
        public void dispC()
            {
                S.o.p("dispC");
            }
    }
public class ClassD extends ClassA
    {
        public void dispD()
            {
                S.o.p("dispD");
            }
    }
public class Test
    {
        public static void main (String args[])
            {
                ClassB b = new ClassB();
                b.dispB();
                b.dispA();
                ClassC c = new class();
                c.dispC();
                c.dispA();
                ClassD d = new classD();
                d.dispD();
                d.dispA();
            }
    }
```

Output - dispB
dispA

disp() C
disp() A
disp() D
disp() A

Multiple and Hybrid Inheritance

Java does not support multiple and Hybrid type of inheritance as the child class has to manage the dependency of more than one parent class. But this can be acheive by using Interface.

Method Overloading and overriding

1. Method Overloading:

When a class has multiple methods having same name but different parameter lists and different definition. This is known as Method Overloading

It is used when objects are required to perform similar task but using different parameter input parameters.

There are two ways to overload a method
- By changing nos. of arguments
- By changing the data type of arguments

It increases the readability of the program.

Example:

```
class Room
  {
    float l;
    float b;
    Room ( float x, float y)
      {
        l = x;
        b = y;
      }
    Room ( float x)
      {
        l = b = x;
      }
    int area ()
      {
        return ( l * b);
      }
  }

class xyz
  {
    public static void main (String args[]).
      {
        Room r = new room(2);
        float area = r.area ();
        S.O.p ( " Area = " + area);
        r.room ( 7, 8 );
        area = r.area();
        S.O.p ("Area = " + area);
      }
  } .
```

Output - Area = 4
Area = 56.

2. Method over-riding

If a subclass contains same method as declared or defined in parent class, it is known as method over riding.

It is used to provide specific implementation of a method that is already provided by a its super class. It is used for runtime polymorphism.

Rules for method Overriding

- method must have same name as in parent class.
- method must have same parameters as in parent class.
- It must posses IS a relationship.
- A static method cannot be Overridden.

Example -

```
class Vehicle
 {
   Void run()
      {
        S.o.p("Vehicle is running");
      }
```

```
class Bike extends Vehicle.
 {
   void run()
     {
       S.o.p("Bike is running safely");
     }

   public static void main(String args[])
     {
       Bike B = new bike();
       B.run();
     }
 }
```

Output - Bike is running safely.

Constructor

Constructor is a special method that is used to initialize the object.
A Java constructor is invoked at the time of object creation. It provides data for the object.

Rules for creating ...

- Constructor name must be same as its class name.
- Constructor must have no explicit return type.
- It is declared as public or protected but not as privat

Example -

```
class Bike
  {
    Bike ()
    {
     S.o.p ("Bike is created");
    }
    public static void main (String args[])
    {
    Bike B= new Bike ();
    }
  }
```

Output:

Bike is created.

Types of Constructor

There are three types of constructor available in java.

1. System supplied constructor
2. Default constructor
3. Parameterized Constructor

1. System Supplied Constructor

In a given class, when we do not write any constructor ourselves, the system supplies a constructor itself. This constructor is known as the System supplied constructor. It does not have any parameters.

- It initializes data numeric data members to zero,

Boolean members to false and String data type with null.

Example.

```
class test
  {
   int a ;
   double b ;      boolean d;
   String  c;
   void show ()
     {
      S.o.p (" int a = " + a);
      S.o.p (" double b = " + b);
      S.o.p (" String c = " + c);
      S.o.p (" Boolean d = " + d);
     }
  }

class test2
   {
    public static void main (String args[])
      {
        test t = new test ();
        S.o.p (" System Supplied Constructor ");
        t.show ();
      }
   }
```

Output- System Supplied Constructor

0

0.0

null

false

2. Default Constructor

If the class has one or more constructors defined, then the system does not supply any constructor. In that case, to initialize the class fields with default values, default constructors are used. In other words, when one or more constructors are present in a class the compiler does not invoke any of them implicitly, the have to be called explicitly through object to initialize the variables with default values.
These kind of constructors are known as the default constructors.

3. Parameterized Constructors

A constructor that have parameters or aggruments is known as parameterized constructor. It is used to provide different values to different objects. We can have any number of paramcters in the constructor.
During Object creation the paramcters we pass determins which constructor should get invoke for object initialization.

Example -

```
class perimeter
 {
  int l;
  int b;
  perimeter ()
  {
   l = 0;          // default Const
   b = 0;
  }
  perimeter (int x, int y)
  {
   l = x;
   b = y;
  }
  void cal ()
   { int peri;
   peri = 2 * (l+b);
   S.O.p("Perimeter=" +peri);
   }
}
Class def
    {
      public static void main (String Args[])
       {
         perimeter p = new perimeter ();   invoke Default
         perimeter p1 = new perimeter (5,10);   invoke param
         p.cal ();
         p1.cal ();
       }
}
```

Output -

```
Perimeter = 0.
Perimeter = 30.
```

Constructor Overloading

It is a technique in which a class can have any

number of constructors that differ in parameter lists. The compiler differentiate these constructors by taking into account the nos. of parameter in the list and their type.

Ex-
```
class Student
 {
   int id;
   String name;
   int age;
   Student (int i, String n)
    {
     id = i ;
     name = n ;
    }
   Student (int i, String n, int j)
    {
     id = i;
     name = n;
     age = j;
    }
   void display()
    {
     S.o.p ( id + " " + name + " " + age);
    }
  public static void main (String args[])
    {
     Student s = new Student ( 1 , "Karan");
     Student s1 = new Student ( 2 , "Vishal" 12);
```

Output - 1 Karan 0
2 Vishal 12

Copy constructor

We can copy the values of one object into another by assigning the objects values to another object

Ex-
```
class Student
 {
  int id ;
  String name;
  Student ( int i, String n)
   {
    id = i;
    name = n;
   }
  Student                    public Student (Student s)
  void display()                 { id = s.id;
   {                               name = s.name;
    S.o.p (id + " " + name);     }
   }
  public static void main (String args[])
    {
     Student s = new Student ( 1, "Karan");
     Student s1 = new Student (s);
     S. display();
     S1. display();
    }
```

Access, modifier abstract class-name
{ abstract methods, data
declare fields
declare method

Abstract Class and Abstract Method

A class which contains abstract keyword in its declaration is known as abstract class.
Abstract classes may or may not contain abstract method i.e method without body.
In java it is used to achieve abstraction i.e hiding the implementation details from the user and providing only functionality.
If a class declared abstract, it can not be instantiated.
To use an abstract class, one have to inherit it from another class, providing implementation to the abstract methods in it.

Abstract Methods

An abstract method is a method that contains a method signature but not method body. Abstract keyword is used to declare the method as abstract.
The actual implementation of an abstraction method is defined in the sub class of the class where the abstract method is present.

An abstract class can have data member, abstract method, method body, constructor and even main method.

Ex-

```
abstract class Bike
 {
   Bike()
   {
    S.o.p ("Bike is created");
   }
   abstract void run();
   void next()
   {
    S.o.p ("Next function");
   }
 }
class Honda extends Bike
 {
  void run()
    {
      S.o.p ("running safely...");
    }
 }
Class Test
 {
   public static void main (String args[])
     {                    Honda
       Bike B = new Bike();
       B.run();
       B.next();
     }
 }
```

Bike is created
running succesfully
Next function

Interfaces

An interface is a mechanism to acheive abstraction and multiple inheritance. It is a blue print of a class and collection of abstract method. It also contain static methods, constants and default methods.
Method in interface do not have a body, it only contain method signature, so methods needs to be implemented by the class.

Defining an interface

Syntax –

```
interface interface_name
{
  // any nos. of public, static and final fields
  // any nos. of static and abstract method.
}
```

Implementing interface

Once an interface is defined, one or more class can implement it.

```
interface example
{
  public void msg1();
  public void msg2();
}
class final1 implements example
{
  public void msg1()
  {
    S.o.p ("Hello");
  }
  public void msg2()
  {
    S.o.p ("Java");
  }
  public static void main (String args[])
  {
    final1 f = new final1()
    f.msg1();
    f.msg2();
  }
```

Interfaces with Multiple Inheritance

```
interface test1
{
  public void method1();
}
interface test2
{
  public void method2();
}
class test3 implements test1, test2
{
  public void method 1()
  {
    S.o.p ("Multiple");
  }
  public void method 2()
  {
    S.o.p ("Inheritance");
  }
```

```
public static void main (String args[])
  {
    Test3 t = new test();
     t.method1();
     t.method2();
   }
}
```

Output - Multiple ~~Inheritance~~
Inheritance

Keywords:

1. Final Keyword.

The ~~the~~ final keyword in java is used to restrict the user. The java final keyword can be used by
- variables
- method
- class

⇒ The final keyword, if used with a variable makes it permanently constant variable we cannot change the value of a final variable

Syntax final return_type variable name = Value;
final int speed = 90;

→ The final keyword is used with a method, then we cannot override. It stops methods overriding functionality in the program.

Syntax - final return type method name()
{
Body
}

eg: final void run()
{
s.o.p ("running");
}

⇒ If a class is declared as final class, then we cannot extend such class. It stops inheritance if it is applied in a program.

Syntax - final class class_name
{
fields / methods.
}

eg - final class Bike
{
-
-
}

2. This:
This keyword in Java is a reference variable that refers to the current object. Following are its usage of this keyword.

1. This keyword can be used to refer current class instance variable

Ex-

```
class test1
  {
   int num=100;
  }
class test2 extends test1
  {
   int num=110;
   void printnumber()
     {
      S.o.p (super.num);
     }
   public static void main (String args[])
     {
      test2 obj = new test2();
      obj.printnumber();
     }
  }
```

Output - 100.

4. Static :

The static keyword is used mainly in java for memory management. It is a keyword that are used for sharing the same variable or method of a given class. This is used for a constant variable or a method that is same for every instance of a class.

It is a non access modifier in java which is applicable for blocks, variables, methods and nested classes precede its declaration with the keyword static.

When any of these members can declared static it can be accessed before any object its class are created and without reference to any object.

→ Syntax to declare a static variable.

public static return type var-name;

→ Syntax to declare static method.

public static void method_name;

→ Syntax for accessing static methods and variable.

classname.variable name = 10;
" . method name();

Ex - Class test
{
Static int a = m1();
Static // to initialize static variables.
{
S.o.p ("Inside static block");
}
static int m1()
{
S.o.p ("from m1");
return 20;
}
public static void main (String args[])
{ So.p ("Value of a :" + a);
S.o.p ("from main");
}
}

from m1
Inside static block
value of a: 20
from main

Java Application

JDBC API

Unit - 4.

Driver Manger

ODBC Driver

Database

JDBC

Java Database Connectivity is an application program-ming interface (API) for programming language Java, which defines how a client may access a database. It is java based data access technology that is part of the Java Standard Edition platform from Oracle corporation. It provides method to query and update data in a database, and is oriented toward relational databases. A JDBC-to-ODBC bridge enables connection to any ODBC accessible data source in the Java Virtual Machine (JVM) host environment.

The JDBC API uses java standard classes and interfaces to connect databases. In order to use a JDBC to connect Java application to a specific database server, a JDBC driver that support the JDBC API for that database server is required

JDBC Drivers

A JDBC driver is a software component enabling a Java application to interact with a database. JDBC drivers are analogous to ODBC drivers ADO.NET, data providers and OLE DB providers.

To connect with individual database, JDBC requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

JDBC Technology drivers fit into one of four categories

1. JDBC-ODBC bridge
2. Native-API driver
3. Network protocol driver (middleware driver)
4. Database Protocol driver (Pure Java driver or thin driver)

1. JDBC-ODBC

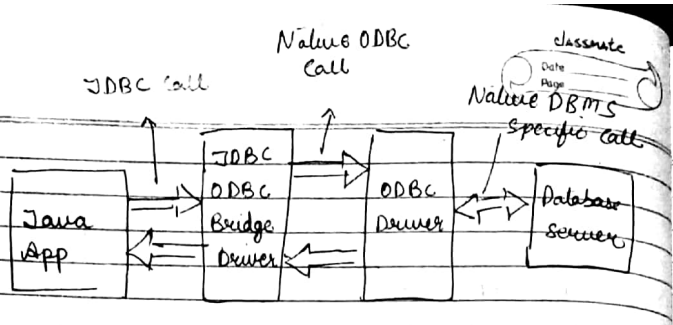
In a type 1, a JDBC-ODBC Bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database. It act as a bridge between JDBC and other database connectivity mechanism. This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.

When java first came out, this was a useful driver because most databases only support ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available

The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Advantages - 1 Easy to use

2. Allow easy connectivity to all databases supported by the ODBC driver.



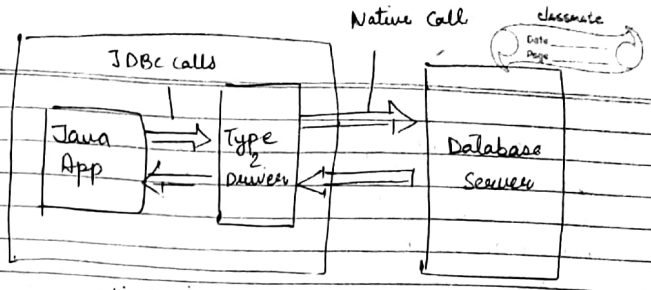
Disadvantage

1. Slow execution time
2. Dependent on ODBC Driver
3. Uses Java Native Interface (JNI) to make ODBC call.

2 Type 2: JDBC - Native API

In a Type 2 driver, JDBC API calls are convert into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some ~~special~~ speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



Advantage

1. faster as compared to Type-1 Driver
2. Contains additional features.

Disadvantages

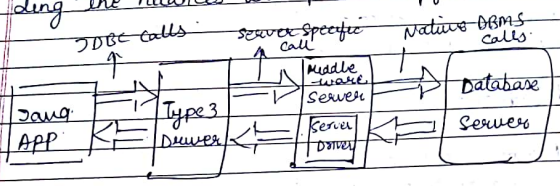
1. Requires native library.
2. Increased cost of Application

3 Type 3 JDBC - Net pure Java

In a type 3 driver, a three tier approach is used to access databases. The JDBC client uses standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to database server. The kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

You can think of the application server as a JDBC "proxy", meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a type 1, 2 or 4 driver to communicate with the database, understanding the nuances will prove helpful.



Advantages:

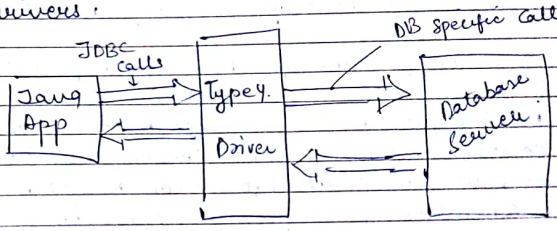
1. Does not require any native library to be installed.
2. Database Independency.
3. Provide facility to switch over from one database to another database.

Disadvantages:

1. Slow due to increase number of network call.

4. Type 4: 100% Pure Java.

In a type 4 driver, a pure Java based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
The kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
MySql's connector/J driver is a type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.



Advantage

1. Does not require any native library.
2. Does not require any middleware library.
3. Better performance than other driver.

Disadvantages

Slow due to increase numbers of network calls

Advantages of JDBC

1. Provide Existing Enterprise Data

Businesses can continue to use their installed database and access information even if it is stored on different database management systems

2. Simplified Enterprise Development

The combination of the Java API and JDBC API makes application development easy and cost effective.

3. Zero configuration for Network Computers

No configuration is required on the client side. Centralizes software maintenance. Driver is written in Java, so all the information needed to make a connection is completely defined by the JDBC URL or by a Data Source object. Data Source object is registered with a Java naming and Directory Interface JNDI naming service.

4. Full Access to Metadata

The underlying facilities and capabilities of a specific database connection need to be understood. The JDBC API provides metadata access that enables the development of sophisticated application

5. No Installation

No pure JDBC technology-based driver does not require special installation.

6. Database Connection Identified by URL

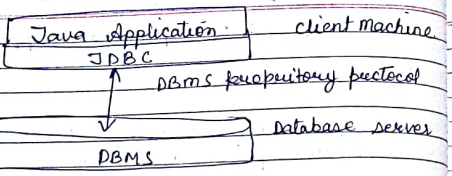
The JDBC API includes a way to identify and connect to a data sourcing, using a Data Sourcing Object. This makes code even more portable and easier to maintain

Two Tier and Three tier client Server model

1. Two Tier Client Server model.

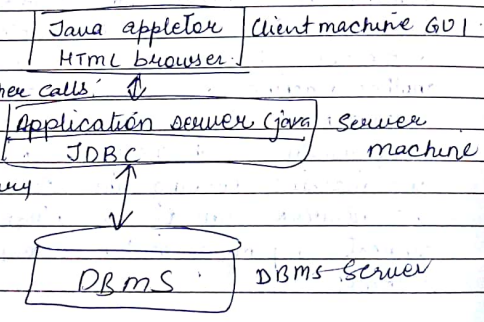
In Two-tier model, a java application talks directly to the data source.
This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the result of those statement are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and machine housing the data source as the server. The network can be an intranet which for ex- connects employees within a corporation or it can be Internet

JDBC supports both two tier and three tier processing models for database access.



2. Three-tier model

In three tier model, commands are sent to a middle tier of service, which then sends the command to the data source. The data source processes the command and sends the result back to the middle tier, which then sends them to the user. MIS directors finds the three tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of application. Finally in many cases the three tier architecture can provide performance advantage.



JDBC - Database Connection

After the installation of the appropriate driver, it is time to establish a database connection using JDBC.
Here are simple steps -

1. Import JDBC packages:

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

```
import java.sql.*;
```

2. Load and Register the JDBC Driver

In this step we load the JDBC driver into JVM. This step is also called as registering the JDBC driver. The forName() method of class Class is used to register the driver class. This method is used to dynamically load the driver class. This step can be completed into two ways:

- Class.forName("fully qualified classfilename")

  Example - Class.forName("oracle.jdbc.driver.OracleDriver");

- DriverManager.registerDriver(object of driver class)

  Example - DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

3. Open a connection to the database

In this step connection between a java program and a database will be opened. To open the connection we call getConnection() method of DriverManager class.

For getConnection() method we need to pass three parameters

- Url
- username
- password

Url is used to select one registered JDBC driver among multiple registered driver by DriverManager class.

Username and passwords are used for authentication purpose.

```
Connection con = new DriverManager.getConnection
                 (url, username, password);
```

Example -

```
Connection con = new DriverManager.getConnection
                 ("Jdbc:Odbc:<dsn>", "scott", "tiger");
```

4. Create a statement object to perform a query.

To transfer sql command from java program to database we need statement object. To create a statement object we can call CreateStatement() method of connection interface. The object of statement is responsible to execute queries with the database.

Example -

```
Statement stmt = new createStatement();
con.createStatement();
```

5. Execute the statement object and return a query ResultSet

executeQuery() method of Statement interface is used to execute a query for select operation executeUpdate for non select operation and execute for both select and non select operations.

These methods return the object of ResultSet that can be used to get all records of a table.

Example:

```
ResultSet rs = o.executeQuery ("Select * from user");
while (rs.next())
  {
    System.out.println (rs.getString(1)+" "+ rs.getString
                        (2));
  }
```

6. Closing the ResultSet and Statement

Once the ResultSet and Statement objects have been used, they must be closed explicitly. This is done by calls to the close() method of the ResultSet and Statement classes. The following code illustrates this

```
rset.close();
sql_stmt.close();
```

7. Closing the connection

The last step is to close the database connection opened in the beginning after ~~the~~ importing the packages and loading the drivers. This is done by a call to the close() method of the connection class.
The following line of code does this.

```
conn.close();
```

Java Utilities

1. Java.lang

This package provides classes that are fundamental to the design of the Java programming language. The most important classes are Object which is the roots of the class hierarchy, and Class, instances of which represent classes at run time.

some

Following are the important classes in Java lang packages.

1. Boolean: The Boolean class wraps a value of the primitive type boolean in an object.

Similarly there are Byte, Character, Double, Integer class that wraps a value of the primitive of that particular type in an object.

2. ClassLoader: A class loader is an object that is responsible for loading classes.

3. Compiler: The Compiler class is provided to support Java-to-native code compilers and related services.

4. Package: Package objects contain version information about the implementation and specification of a Java package.

5. Runtime: Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running

2. Java.util

This package contain the collections framework, legacy collection classes, event model, date and time facilities, internationalization and miscellaneous utility classes. Following are some important classes

in Java.util package.

1. AbstractionCollection: This class provides a skeleton implementation of the Collection interface to minimize the effort required to implement this interface.
2. Array: This class contains various methods for manipulating arrays.
3. Currency: It represents a currency.
4. Date: The class date represent a specific instant in time with millisecond precision.
5. Properties: This class represents a persistent set of properties.

3. Java.io

This package provides for system input and output through data streams serialization and the file system. Unless otherwise noted, passing a null argument to a constructor or method in any class or interface in this package will cause a NullPointerException to be thrown.
Java encapsulates Stream under java.io packages
Java defines two types of Stream. They are

1. Byte Stream
2. Character Stream

1. Byte Stream: It provides a convinient means for handling input and output of byte.
2. Character Stream: It provides a convinient means for handling input and output of characters. Characters Stream uses Unicode and therefore can be internationalized.

Unit - 3

Applets

Java applet inherits features from the class Applets are small ~~java~~ application that can be accessed ~~on an internet server~~

Applet Java

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser ~~browser~~ and works at client side. They can be transported over the internet from one computer to another and run using the Applet Viewer or any Web Browser that support Java. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation and play interactive games.

Example program:

```
import java.applet.*;
import java.awt.*;

public class HelloWorldApplet extends Applet
{
    public void paint (Graphic g)
    {
        g.drawString("Hello World", 25, 50);
    }
}
```

Applet life cycle

Following are the various stages of an applet life cycle:

1. Initial state Born State

When an applet is born or created, it is activated by calling init() method. At this stage, new objects to the applet are created, initial values are set, images are loaded and the colors of the image are set. An applet is initialized only once ~~in~~ its life time. Its general form is:

```
public void init ()
{
    // Action to be performed
}
```

2. Running state

An applet acheives the running state when the system calls the start() method. This occurs as soon as the applet is initialized. An applet may also start when it is in idle state. At that time, the start() method is overridden. Its general form is:

```
public void start ()
{
// Action to be performed
}
```

3. Idle State

An applet comes in idle state when its execution has been stopped either implicitly or explicitly. An applet is implicitly stopped when we leave the page containing currently running applet. An applet is explicitly stopped when we call stop() method to stop() it execution. It general form is

```
public void stop()
{
// Action to be performed.
}
```

4. Dead State

An applet is in dead state when it has been removed from the memory. This can be done by using destroy() method. Its general form is:

```
public void destroy()
{
// Action to be performed.
}
```

Apart from the above stages, Java applet also possess paint() method.

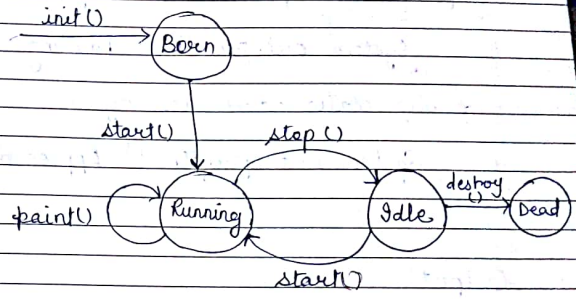
```
paint()
```

This method helps in drawing, writing, and creating coloured backgrounds of the applets.

It takes an argument of the graphics class. To use the graphics, it imports the package java.awt.Graphics.

The method execution sequence when an applet is executed :
→1. init()
→2. start()
→3. paint()

The method execution sequence when an applet is closed →1. stop()
2. destroy()



Example - demonstration of the life cycle of applet

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
    public void init()
    {
```

```
    System.out.println("Applet initialized");
}
public void start()
{
    System.out.println("Applet execution started");
}
public void stop()
{
    System.out.println("Execution stopped");
}
public void paint(Graphic g)
{
    System.out.println("Painting...");
}
public static void destroy()
{
    System.out.println("Applet destroyed");
}
}
```

Output:

Applet initialized.
Applet execution started
Painting...
Painting...
Execution stopped
Applet destroyed

Applet vs application program

Sno	Comparision Criteria	Applet	Application
1.	Basic	It is small program uses another application program for its execution	An application is the programs executed on the computer independently.
2.	main method	It does not use the main method.	Uses the main method for execution
3.	Execution	It cannot run independently. It requires API's Ex web Api	Can run alone but require JRE, Java Runtime Environment
4.	Installation	Prior installation is not needed.	Requires prior explicit installation on the local computer
5.	Read and Write operation	Applets can not read from and write to the files on the local computer.	Application are capable of performing these operation to the files on the local computer.
6.	Communication with other servers	Can not communicate with other servers	Communication with other servers is probably possible.

7.	Restriction	Applet cannot access files residing on the local computer.	Can access any data or file available on the system.
8.	Security	Requires security for the system as they are untrusted	No security concerns are there.

AWT Event handling

Event :

Change in the state of an object is known as event i.e event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface. For ex.
clicking on a button, moving the mouse etc.

Types of Events.

The events can be broadly classified into two categories

1. Foreground event

These events which requires the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in GUI. Ex clicking on button, moving the mouse etc.

2. Background Event -

Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure etc.

Event Handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java uses the delegation event model to handle the events. This model defines the standard mechanism to generate and handle the events.

Steps involved in event handling

1. The user clicked the button and the event is generated.
2. Now the object of concerned event is created automatically and information about the source and the event get populated within same object.
3. Event object is forwarded to method of registered listen class.

this method is now get executed and returns.

Components of event Handling

1. Event: An event is a change in state of an object
2. Event Source: It is an object that generates an event
3. Listeners: It is an object that listens to the event

A source generates an event and send it to one or more listeners registered with the source

Once event is ~~registered~~ received by the listener they process the event and then return.

Events are supported by a number of Java packages like java.util, java.awt and java.awt.event

4. ~~Delegation Of event model~~

Unit - 2.

Packages

A package in java is a mechanism to encapsulate a group of classes, sub packages and interfaces. In other words we can say packages are java's way of grouping of a variety of classes and/or interface The grouping is usually done according to the functionality. It act as a container for classes.

Packages are used for:

1. Preventing naming conflicts, two classes with the same name in different packages.
2. Making searching of classes, interface etc easier (locating, usage)
3. It provide controlled ~~acc~~ access with the help of access specifiers.
4. To acheive data encapsulation or data hiding

Packages in java can be catagorised in two forms.

1. Built-in packages or Java API packages - These are the large number of classes grouped in different packages according to similar functionality
2. User defined package that are specificly defined by the user.

✱ Package Creation in Java.

The package keyword is used to create package in java.

We must first declare the name of package using package keyword.

Then the classes are defined that are to be included in the package

Syntax

```
package package_name; //package declaration
public class class_name  // class definition
{
    Body of class.
}
```

Example

```
package mypack;
    public class Simple
        {
        public static void main (String args[])
            {
            System.out.println ("Welcome");
            }
        }
```

Creating a package involves the following steps

1. Declare the package at the begining of the file using
   package packagename;
2. Define the class that is to be put in the package and declare it public.
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates .file in the subdirectory.

✱ Compile the package.

Syntax - javac -d. directory javafile_name

Eg - javac -d. simple.java

The -d switches specifies the destination where to put the generated class file. You can use any directory ex d:/abc java. If you want to keep the package in the same directory (.) dot is used.

✱ To run a java package program

To compile : javac -d. simple.java;
To run : java mypack.Simple;

How to access package from another package.

There are three ways to access a package from the outside of the package.

1. import package.*;
2. import package.class-name;
3. fully qualified name;

1. To import all the classes from a particular package.

If we use 'package_name.*;' then all the classes and interface will be accessible but classes and interface inside the subpackage will not be accessible for use

The import keyword is used to make the classes and interface of another package accessible to the current package.

Ex- // save by first.java

```
package learnjava;
public class First
{
    public void msg()
    {
        System.out.println("Hello");
    }
}
```

// save file by second.java

```
package java;
import learnjava.*;
class Second
{
    public static void main(String[] args)
    {
        First obj = new First();
        obj.msg();
    }
}
```

Output: Hello.

2. To import specific class ~~or classes~~ from the package

If we require only ~~some~~ specific class to be used in our program from another package. Then only specified classes and interfaces will be imported from the package and other classes and interfaces in that particular will be unaccessible.

Example

// Save file by A.java

```
package pack;
class public class A
{
    public void msg()
    {
        System.out.println("Hello");
    }
}
```

// Saved as B.java

```
package mypack;
import pack.A;
public class B.
    {
        public static void main (String[] args)
            {
                A obj = new A();
                obj.msg();
            }
    }
```

Output: Hello.

(2) Using fully qualified name.

The another approach to use the method of a class present in an another package is to define the fully qualified name of package and class that we want to use with at the creation of the object.
We need to use fully qualified name every time in the program for accessing the class or interface which look like very untidy if the package is very long.

Example:

// save by file A.java

```
package pack;
public class A
    {
        Public void msg()
            {
                public System.out.println("Hello");
            }
    }
```

// Save as B.java

```
package mypack;
public class B.
    {
        public static void main (String args[])
            {
                pack.A obj = new pack.A();
                obj.msg();
            }
    }
```

Output ! Hello.

As we can see while using fully qualified name we don't need to use import keyword.

→ Import Keyword - Import keyword is used to import build in and user defined packages into your java source file, so that a class can refer to class that is present in another package by directly using its name.

→ Static Keyword · import

Static keyword is a feature introduced in Java programming language (Version 5 and above) that allows members (fields and members) defined in a class as public static to be used in java code without specifying the class in the field is defined.

To provide the accessibility to the user any static member of a class directly.

Advantage:

1) Less coding is required if you have access to any static member of a class oftenly.

Disadvantage:

It overuses the static import feature, it makes the program unreadable and unmaintainable.

Example

```
import static java.lang.System.*;
class StaticImportExample
{
    public static void main (String args[])
    {
        out.println ("Hello");
        out.println ("Java");
    }
}
```

Output : Hello
Java.

Subpackages:

Packages that are inside another packages are called subpackages. These are not imported by default they have to imported explicitly. Also the members of a subpackages have no access privilege i.e they are considered as different for protected and default access specifier.

Ex- import java.util.*;
util is a subpackage create inside a java package.

```
package com.javapoint.core;
class Simple
{
    public static void main (String Args[]),
    {
        System out.println ("Hello subpackage");
    }
}
```

* Transient Keyword.

Transient is a keyword which marks a member variable not to be serialized when it is persisted its stream of bytes. When an object is transferred through the network, the object needs to be serialized.
Serialization converts the object state to serial byte. These byte are sent over the network and the object is recreated from those bytes. Member variable marked by java transient keyword are not transferred, they are lost intentionally.
It plays an important role to meet security constraints.
It is used not to serialize variable whose value can be calculated / derived using other serialized objects such as age of person, current date.

Syntax

private transient <member variable>;
or
transient private < member variable>;

For example, a class student is declared, it has three data members id, name and age we dont want to serialize one value i.e age then we can declare the age data member as transient.

8. Access Specifies / Modifiers

In java the term access modifier refers to the keywords which are used to control the accessibility to class, interfaces, fields, constructors and methods. That means we can determine what can access our code. These are used to protect the data and behaviours from from the outside world. They are the core fundamental and important concepts in java which requires its grasp a deep understanding in order to write code properly, efficiently, surely and professionally.

Types of access modifiers in java are as follows.

1. Private
2. Default
3. Protected
4. Public

Private Access Modifier

The private access modifier is accessible only within class. The scope of private modifier is limited to the class only in which it is declared. Private data members and method are only accessible within class. These cannot be accessed by any other classes. Classes and Interface cannot be declared as private in java. If a class has private constructor then you can create the object of that class from outside the class. The keyword private is used to declare classes, members and functions

```
package p1;
Class A
    {
    private void display ()
        {
        System.out.println ("Hello");
        }
    }
Class B
    {
    public static void main (String args[])
        {
        public static void main (String
        A obj = new A ();
        Obj.display ();
        }
    }
```

Output → error: display() has private access in A obj.display()

2) Default Access Modifier

When no access modifier is specified in a class, method or data member is said to be having the default access modifier by default.

The data members, class or methods which are specified as default or not specified with any specifier will be accessible within the package

Ex- In this example, we will create two packages and the classes in the package will be having the default access modifier and we will try to access a class from one package from a class of second package.

```
package p1
  class Geek
      {
      void display ()
          {
          System.out.println ("Hello");
          }

package p2
  import p1.*;
  class geekNew
      {
      public static void main (String args[])
          {
          Geek obj = new Geek ();
          obj.display ()
          }
      }
```

Output: Compile time error

This is the method display () of package p1 class geek cannot be accessed by another class because they are declared as default access modifier that have range of using the method with the package in which they are declared

③ Protected Access Modifier

The protected access modifier is specified using the keyword protected. The method declared as protected are accessible within the same package or by the subclass in different package. Classes cannot be declared as protected in java. This access modifier is generally used in a parent child relationship.

Ex In this example we will create two packages p1 and p2. Class A in p1 ~~and~~ is made public to access it in p2. The method display in class A is protected and class B is inherited from class A and this protected method is then accessed by creating an object of B.

```
package p1;
public class A
    {
        protected void display()
            {
                System.out.println("Hello World");
            }
    }
package p2;
import p1.*;
class B extends A
    {
    public static void main(String args[])
        {
            B obj = new B();
            obj.display
```

Output - Hello World.

④ Public Access Modifier

The public access modifier is specified using the keyword public. The scope of a public access modifier is the widest among all other access specifier. Classes, methods or data members which are declared as public are accessible from everywhere in the program. There is no restriction on the scope of a public data members.

```
// java program to illustrate public modifier
package p1
public class A
    {
        public void display()
            {
                System.out.println("Hello");
            }
    }
package p2
import p1.*;
class B
    {
        public static void main(String args[])
            {
                A obj = new A();
                obj.display();
            }
    }
```

Output: Hello

Volatile Keyword.

Volatile keyword in java is used as an indicator to java compiler and thread that do not cache value of the variable and always read it from the main memory. It cannot be used be used with methods or class and it can be used only with variable. If the variable keeps changing such time of variables we have to declare with volatile modifier. If a variable declared as volatile then for every thread a separate local copy will be created.

Example -

```
public class Myclass
{
    private volatile int value;
    public int getvalue()
    {
        return value;
    }
    public void setvalue (int value)
    {
        this.value = value;
    }
}
```

an event disrupt the normal flow of the program

* Exception Handling

Exception Handling is the mechanism to handle runtime malfunctions. We need to handle such exceptions to prevent abrupt termination of program. The term exception means exceptional conditions, it is a problem that may arise during the execution of program a bunch of things can lead to exceptions including programmer error, hardware failure, files that need to opened cannot be found, resource exhaustion etc.

Exception : A java exception is an object that describes the exception that occurs in a program. When an exceptional event occurs in java, an exception is said to be thrown. The code that is responsible for doing something about the exception is called an exception handler.

* Error vs Exception

1. Error : An error indicates a serious problem that a reasonal application should not try to catch.
2. Exception: Exception indicates conditions that a reasonable application might try and catch.

* Types of Exception

1. Checked Exception

The exception that can be predicted by the programm

at the compile time are called checked exception Ex - File that need to be opened not found, these types of exceptions must be checked at compile time.

2 Unchecked Exceptions

Unchecked exception are the class that extends runtime exception. Unchecked exceptions are ignored at compile-time. Ex. Arithmetic Exception, Null pointer exception, Array index out of bound etc Unchecked exceptions are checked at runtime. Exception those occurs due to programming bugs or improper API. These are called Runtime exception

3. Error -

Error are typically ignored in code because we can rarely do anything about an error. Ex - If stack overflow occurs, an error will arise This type of error cannot be handled in the code.

4 Arithmetic Exception.

It is thrown when an exceptional condition has occurred in an arithmetic operator.

5 Array Index Out of Bound Exception -

It is shown to indicate that an array has been accessed with an illegal index. This index is either negative or greater than or equal to the size of array.

I/O Exception -
When input/output failed or interrupted

Runtime Exceptions

6 Null Pointer Exception -

This exception is raised when referring to the member of a null object. Null means nothing

7 Runtime Exception - occurs during runtime.

8 No Such Method Exception

It is thrown when accessing a method which is not found.

9 File Not Found Exception

This exception is raised when a file is not accessible or does not open.

10 Class Not Found Exception

This exception is raised when we try to access a class whose definition is not found.

11 Not Such Field Exception

It is thrown when a class does not contain the field or variable specified.

12 Number Format Exception

This exception is raised when a method could not

convert a string into a numeric format

13 StringIndexOutOfBoundException

It is thrown by String class method to indicate that an index is either negative than size of String

8 Exceptions Handling Mechanism

In java exception handling is done using five keywords:

1 try
2 catch
3 throw
4 throws
5 finally

1 Try:

The try block contain set of statement where an exception can occur. Try block must be followed by either catch block of finally block.

Syntax: try
{
statement that may cause an exception
}

2 Catch

Java catch block is used to handle the exception It ~~can occur~~ must be used after the try block only. We can use multiple catch block with a single try.

Syntax: try
{
// Statement that may cause exception
}
Catch (exception (type) e (object))
{
// error handling code
}

Implementation

Try is used to guard a block of code in which exception occur. This block of code is called guarded region A catch statement involves declaring the type of exception you are trying to reach to catch. If an exception occurs in guarded region, the catch block that follows the try is checked, if the type of ~~occur~~ exception that occurred is listed in the catch block, then the exception is handled over to catch block which then handles it.

Ex-
class Excp
{
public static void main (String [] args)
{
int a, b, c;

```
try {
    a = 0
    b = 10;
    c = b/a;
    System.out.println("This will not be executed");
}
catch (ArithmeticException e)
    {
    System.out.println("Divide by zero");
    }
System.out.println("After exception is handled");
}
```

Output - Divide by zero.
After Exception is handled.

Explanation

An exception will be thrown by this program as we trying its divide number by zero inside try block. The program control is transferred outside the try block. Thus the line "This line will not be executed" is never parsed by a compiler. Once exception is handled the program control is continue with the next line in the program i.e after catch block. Thus, the line 'After exception handled" is printed

Multiple Catch block.

A try block can be followed by multiple catch block. We can have any number of catch block after a single block. If an exception is passed to first catch block list and it does not matches with it, then again it will be matched with the next catch block. This will continue until the exception is caught or falls through all catches.

```
Ex- class Excep
    {
    public static void main (String [] args)
    {
    try
    {
    int arr[] = {1, 2};
    arr[2] = 3/0;
    }
catch (Arithmetic Exception e)
    {
    System.out.println("divide by zero");
    } Array
catch (IndexOutOfBoundException e)
    {
    System.out.println("array index out of
                        bound");
    }
    }
}
```

Output: divide by zero.

③ Finally

A finally keyword is used to create a block of code that follows a try block. A finally block of code always executed whether an execution has occured or not. Using finally, let us run any cleanup type statement that user want to execute, no matter what happens in the protected code. It appears at the end of catch block.

Syntax :

```
try
{
// statement that may cause an exception
}
catch
{
// handling exception
}
finally
{
// statement to be executed
}
```

Ex -

```
class example
{
public static void main (String args[])
{
try
{
int num = 121/0;
System.out.println (num);
}
catch (Arithmetic Exception e)
{
System.out.println ("Divide by zero");
}
finally
{
System.out.println ("This is finally block");
}
System.out.println ("Out of try, catch, finally");
}}
```

Output - Divide by zero
This is finally block.
Out of try, catch, finally

④ Throw -

The throw keyword in java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exception.

Syntax - throw Throwable Instance

Example - throw new Arithmetic Exception ("1by zero")
Instance must be throwable or subclass of throwable
For eg: exception is a subclass of throwable.

Exp-

```
class ThrowExcep
{
    Static void fun ()
    {
        try
        {
        throw new NullPointerException ("demo");
        }
        Catch (NullPointerException e)
        {
        System.out.println(" Caught inside fun()");
        throw e;
    }}
    public Static void main (String args[])
    {
        try
        {
        fun();
        }
        catch (NullPointerException e)
        {
        System.out.println ("Caught in main");
        }
    }
```

Output - Caught inside fun()
Caught in main

(5) Throws :

Throws is a keyword in Java which is used in the signature of method. It indicate that this method might throws one of the listed type of exception. The calls to these methods has to handle the exception using a try - catch block. This block is used to declare an exception, which means it works similar to the try catch block.

Syntax:

```
Type method name (parameter list) throws exception
{
    // definition method.
}
```

Example : demonstrating throw and throws keyword.

```
Class Test
{
    Static void check() throws ArithmeticException
    {
        S.o.p("Inside check function");
        throw new ArithmeticException("demo");
    }
    public Static void main (String args[])
    {
        try
        {
            check()
        }
        Catch (ArithmeticException e)
```

```
    {
    S.o.p ("Caught" +e);
    }
  }
}
```

Output: Inside check function
Caught: java.lang.ArithmeticException: demo

Throw Vs Throws -

Basis	Throw	Throws
Keyword	Throw keyword is used to throw an exception explicitly.	Throws keyword is used to declare an exception possible during execution.
Following	Throw keyword is followed by an instance of Throwable class or one of its subclasses.	Throws keyword is followed by one or more exception class name separated by commas.
Declaration	Throw keyword is declared inside method body.	Throws keyword is followed by one or more Exceptions names separated by commas and used with method signature (method declaration)
Multiple exception	We can not throw multiple exceptions using throw keyword.	We can declare multiple exceptions separated by commas using throws keyword.
Syntax	Throw ThrowableInstance	type methodname(para) throws exceptionlist

Rewording in Java:
User defined Exception

Can be made by using throw and extending exception class.

```
class JavaException
  {
   public static void main (String args[]).
    {
     try
      {
       throw new Myexception(2);
    //throw is used to create new exception.
      }
       catch (myexception e)
        {
         S.o.p(e);
        }
  } }
class myexception extends Exception
 { int a;
   My Exception (int b)
```

{
a = b;
}
public String toString()
{
return ("Exception Number = " + a);
}
}
Exception Number = 2

Threading

Threading in Java.

Thread

A thread in the context of Java is the path followed when executing a program. All java programs have at least one thread known as the main thread, which is created by JVM at the program's beginning, when the main() method is invoked with the main thread. In java, creating a thread is accomplished by implementing an interface and extending a class. Every java thread is created and controlled by the java.lang.Thread class.

It is basically a light weight sub process, a smallest unit of processing. Threads are independent, if there occurs an Exception in one thread, it does not affect other threads. It shares a common memory area.

Creating a thread

Java defines two ways by which a thread can be created.

→ By implementing the runnable Interface.
→ By extending thread class.

Implementing the runnable Interface

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface, the class needs to implement the run() method is of form,

public void run()

→ Run() method introduces a concurrent thread into the program. The thread will end when run() method terminate.

→ We must specify the code that the thread will execute inside run() method.

→ run() method can call other methods, can use other classes and declare just like any other normal method.

Example:

class myThread implements Runnable.
{
public void run()
{

```
        S.o.p ("Concurrent thread started running");
    }
}
class MyThreadDemo
{
    public static void main (String args[])
    {
        MyThread mt = new MyThread ();
        Thread t = new Thread (mt);
        t.start ();
    }
}
```

Output: Concurrent thread starts Running

To call the run method start() method is used. On calling start() a new stack is provided to the thread and run() method is called to introduce the new thread into the program

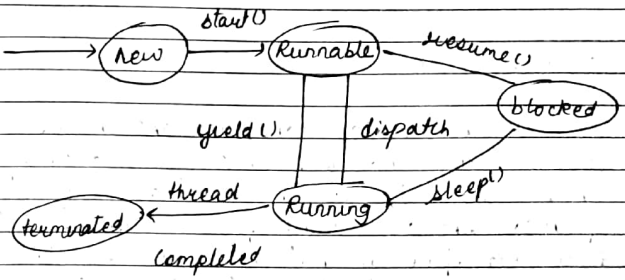
② Extend thread Class.

This is another way to create a thread by a new class that extends Threads class and create instance of the class. The extending class method must override run() method which is the entry point of new thread.

```
class MyThread extends Thread
{
    public void run()
    {
        S.o.p ("Start running");
    }
}
class MyThreadDemo
{
    public static void main (String args[])
    {
        MyThread mt = new MyThread();
        mt.start();
    }
}
```

Output : Start running

* Life Cycle of Thread



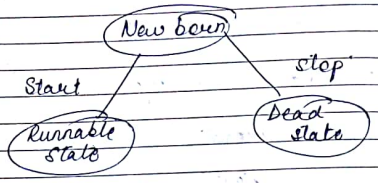
During the life cycle of a thread, there are many states it can enter. They are

1. New born state

When we create a thread object, the thread is born and is said to be in new born state. The thread is not ~~full~~ yet scheduled for running. At this state we can do only one of the following things

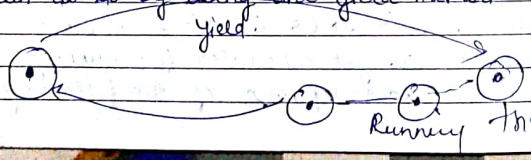
→ Schedule it for running using start() method.
→ Kill it using stop() method.

If scheduled it moves to the runnable state. If we attempt its use any other method at this stage an exception will thrown.



(2) Runnable State

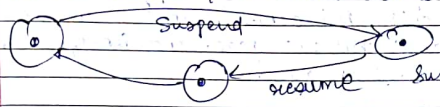
The runnable state means that the thread is ready for execution and is waiting for availability of the processor. That is, the thread has join the queue of thread that are waiting for execution. If all threads have equal priority then they give time slots for execution in round robin fashion i.e first come first serve manner. The thread that relinquish control joins the queque at the end and again wait for its turn. This process of assigning time to thread is known as time slicing. However if we want a thread to relinquish control to another method to equal priority before its turns comes, We can do so by using the yield method.



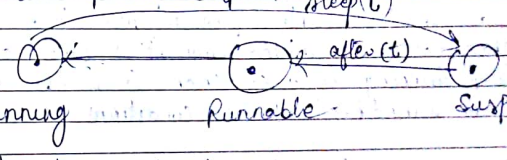
Running State

Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquish control on its own or it is preempted by a higher priority thread. A running thread may relinquish its control in one of the following situation

(1) It has been suspended using suspend() method. A suspended thread can be revived by using the resume() method. This approach is useful when we want to suspend a thread for sometime due to certain reason, but do not kill it

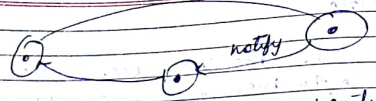


(2) It has been made to sleep. We can put a thread to sleep for a specific time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of queue during this time period and re-enter as soon as this time period elapsed



(3) It has been said to wait until some event occurs. This is done by using wait() method. The thread can be scheduled to run again using the notify method

wait



Running Runable waiting

(4) Blocked state

A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping or waiting in order to satisfy certains requirements. A blocked thread is considered not runnable but not dead and therefore fully qualified to run again.

(5) Dead State

Every thread has a life cycle. A running thread ends its life has complete executing its run() method. It is a natural death. However we can kill it by sending the stop message to it at any state thus causing a pre-mature death to it. A thread can be killed soon it is born or while it is running an even when it is not in "non runnable" blocked conditions

Methods

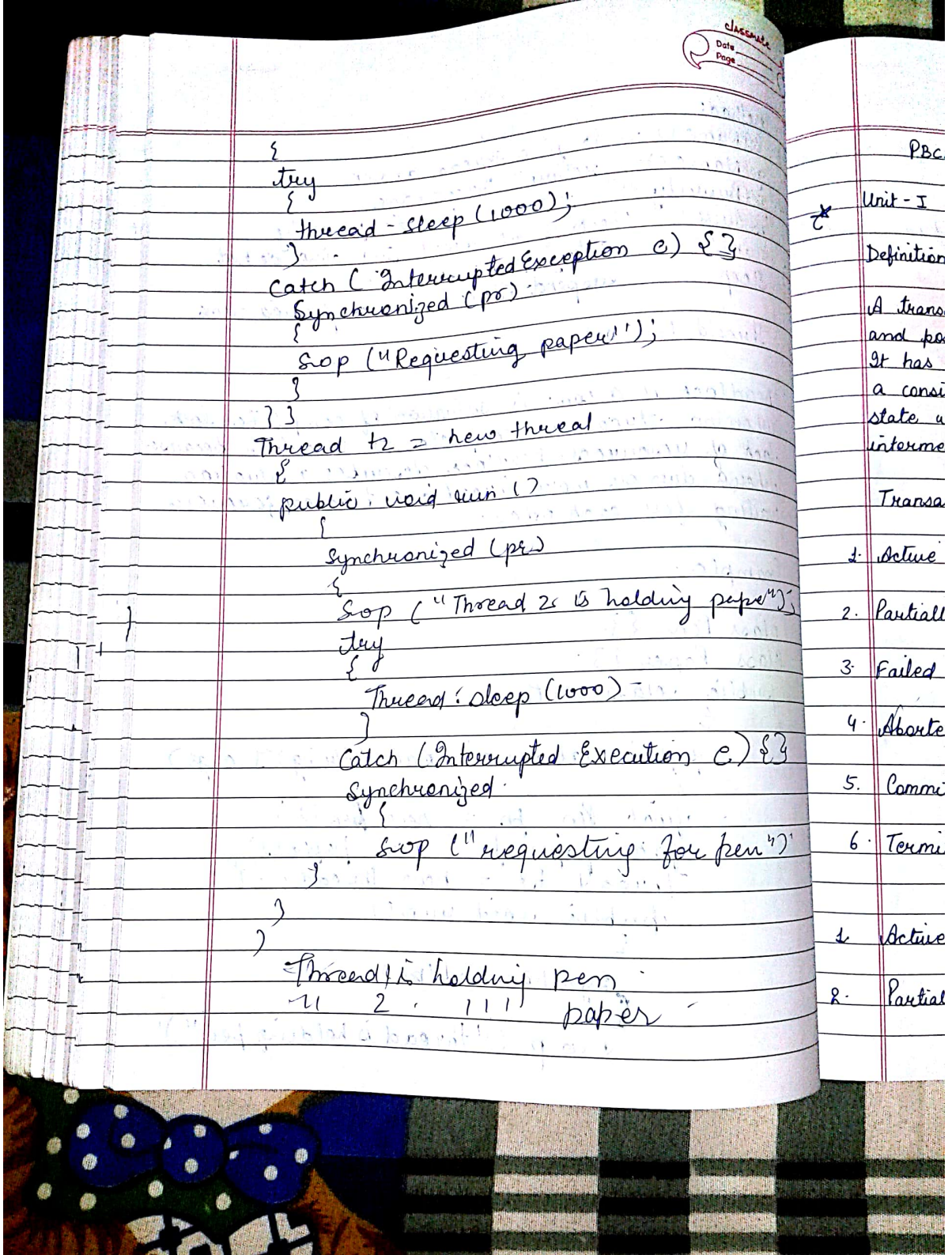
1. SetName() = to give thread a name
2. getName() = returns thread name
3. getPriority() = returns thread's priority
4. isAlive() = Checks if the thread still alive or not
5. run() = Entry point for a thread.
6. sleep = suspend thread for a specified time

8 Thread Deadlock

Deadlock is a ~~com~~ is situation of complete lock. when no thread can complete its execution because lack of resources. Deadlock describes a situation where two or more thread are blocked forever, waiting for each other.

Example.

```
class Pen {}
Class Paper {}
public class write
    {
        public static void main (String[] args)
        {
        final Pen pn = new Pen();
        final paper pn = new paper();
        Thread t1 = new thread() {
        public void run()
        {
            Synchronized (pn)
            {
            s.o.p ("thread is holding pen");
```



{
try
{
thread.sleep(1000);
}
catch (InterruptedException e) {}
Synchronized (pr)
{
Sop ("Requesting paper");
}
}}
Thread t2 = new thread
{
public void run()
{
Synchronized (pr)
{
Sop ("Thread 2 is holding paper");
try
{
Thread.sleep(1000)
}
Catch (Interrupted Execution e) {}
Synchronized
{
Sop ("requesting for pen")
}
}
}
Thread 1 is holding pen
" 2 " " paper

PBC

Unit - I

Definition

A trans
and po
It has
a consi
state u
interme

Transa

1. Active
2. Partiall
3. Failed
4. Aborte
5. Commi
6. Termi

1. Active
2. Partial