JAVA

Java is technology which is used to develop different types of applications like Desktop, distributed etc;

Desktop Applications: These Applications will Runs on local Machine, These Applications will not runs on both Browser and server and we can not share any information through the internet.

Ex: MS-office, local calculator, VLC Mediaplayer, KM player etc

Distributed Applications: These Applications will runs on both browser and server and we can share any information through the internet.

Ex: Gmail, facebook, Internet Banking etc,

JAVA Software was developed by Sun Microsystems in the year 1994 but released into industry in the year 1995. The father of Java Software is James Gosling.

Now-a-days Java Software acquired by Oracle Corporation.

Java Software is mainly classified into three modules like as follows.

1. JSE (Java Standard Edition).
2. JEE (Java Enterprise Edition).
3. JME (Java Mobile/Micro Edition).

* By using JSE Module we can develop desktop Applications
* By using JEE module we can develop distributed Applications.
* By using JME module we can develop Mobile Applications like Android, Software.

Java Features:

* Simple
* Platform Independent
* Architectural Neutral.
* Portable
* Dynamic
* Distributed
* Networked
* Multi-threading
* High Performance
* Highly interrupted
* Robust
* Secured
* Object oriented programming language.

Simple: Java is Simple programming language because of two reasons

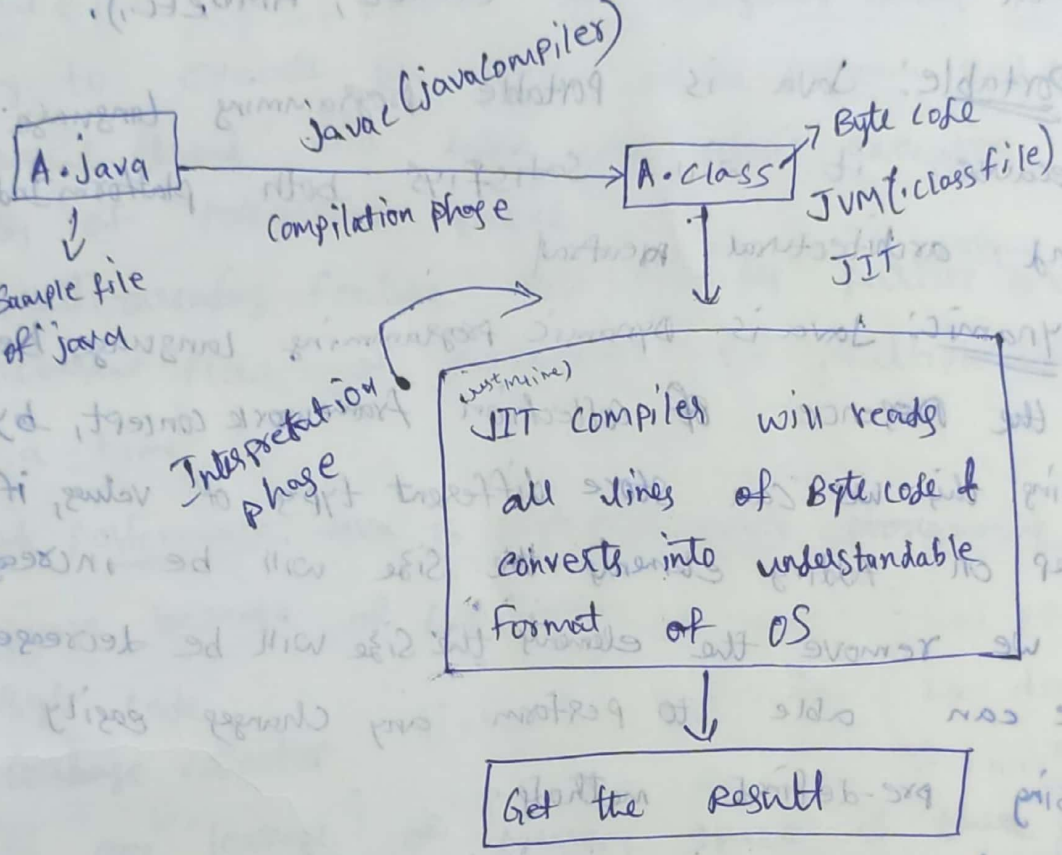
1. Generic API (Application Programming Interface)
2. Byte code.

Generic API: Sun MicroSystems as provided in a facility like API, API is the collection of packages, A package is the collection of classes, Interfaces and Sub packages.

These Multiple classes and Interfaces internally contains pre-defined variables, methods and constructors.

By using API only we can develop different types of Applications.

Byte code:



Platform Independent:

Java is platform Independent programming language why because of following reason:

* Suppose we have two operating systems Window & Linux, we will create java file, we will compile then we will get .class file in window OS, we will copy .class file and then paste in Linux OS, we will run the program there itself and then we will get the result.

* Because of these Java satisfies the slogan like WORA (Write Once Run Anywhere).
* Byte code is platform Independent and JVM is Platform dependent.

Architectural Neutral: Java Software will runs on each and every processor (INTEL, AMD etc.).

Portable: Java is Portable programming language why because, it will satisfiys both platform Independent and architectural neutral.

Dynamic: Java is Dynamic programming language because of the presence of collection framework concept, by using this we can store different types of values, if we keep on adding elements the size will be increases, if we remove the elements the size will be decreases and we can able to perform any changes easily by using pre-defined methods.

Distributed: Java is distributed programming language why because we can able to develop the applications where we can run on both Browser and Server, we can distribute the services (or) functionalities.

Networked: Java is networked programming language why because we can able to develop the applications like client to server, server to database, client to multiple servlets and multiple servlets to multiple databases by using network (IP address, port number, DNS,

URL, PROTOCOL etc.) acts as a communication layer.

Multi Threading: Java is Multi threading programming language why because in each and every program of Java we have two threads ① foreground thread ② Background thread. * The main purpose of foreground thread is going to execute the logic of the program, and the Background thread will takes care about execution status of foreground thread.

* Multi threading feature will provides parallel (or) concurrent flow of execution of Multiple threads at a time.

High performance: Java is High performance programming language because of two reasons

① Byte code

② Garbage collector.

If any leakage of memory space is there then JVM will calls automatically garbage collector and then it will deallocates the memory spaces.

Java software will takes less application development time and less execution time.

Highly Intrepreted: Java is highly Intrepreted because of both JVM and JIT compiler.

Robust: Robust means strong and Java is Robust because of the presence of exception handling, we can able to handle the exceptions by using the keywords like try, catch, finally

throws and throw.

* By using Exception Handling we can able to make the program flow of execution normally without any Interruptions.

Secured: Java is secured programming language why because whenever we transfer the data from client to the server through the network then the data will be transfered in the form of Encrypted format.

Object Oriented Programming Language:

Java is Object Oriented Programming language because of following reasons

* By using Java Software we can able to satisfy each and every OOPS concepts.
* Even if we are not going to create any object for our class, JVM will create it's own object for super class for all the classes is java.lang.Object, these object will acts as memory space for our class properties.
* In our class after creating variables, methods, constructors and object then each and every properties of class available in terms of object.
* By using Wrapper classes we can able to convert primitive type of values into object. or Object type of values into primitive type of data.

Data types: The main purpose of Data type is going to allocate a particular amount of memory space in main memory for the respective value and it will identifies what type of value we need to store.

Classification of data types:

1. primitive or fundamental data types.
2. Non primitive or Derived data types
3. User defined or Custom defined data types.

By using primitive or fundamental data types we can able to store only one value in a single variable but not more

Eg: int a=29; ✓ (valid)
int b= 7, 96 ;x(invalid)

By using derived data types we can store multiple values of similar data type in a single variable but we cannot store multiple values of different data types; once after declaring the size we cannot change on runtime why because arrays will follows static memory allocation

```
int a[] = {2, 3};
int b[] = new int[3];
b[0] = 23;
b[2] = 6;
b[3] = 92; x
int c[] = {32, 'y', 1.69}; x
```

By using custom defined data types we can store multiple values of different data types in a single variable.

To deals with custom defined data types in 'c' language we have a concept called data structures.

Eg: Struct Student {
int stId;
Char st Name [20];
float stMarks;
}

In the case of Java we have both classes and Interfaces to deals with user defined data types.

Eg: class Employee
{
int empId;
String empName;
double empSalary;
}

Primitive or Fundamental data types are classified into following types

1. Integer category data types
2. float category data types
3. Character category data types
4. Boolean category data types

By using Integer category data types we can store only numerical values and we have the data types like byte, Short, int, long.

By using float category data types we can able to store decimal type of values and numbers also and we have the data types like float & double

By using character data type we can able to store single character type of values and we need to specify the characters with in single quo and we have the data type like char.

By using Boolean category data type we can able to store true or false type of values and we have the data type like Boolean.

Variables:

It is an Identifier which is used to Identify or recognise a respective value going to be stored in main memory and whose value will be varies (or) modifies from one point of time to another point of time.

Types of variables:

1. Instance (or) Non-static variables.
2. Static variables.
3. local variables.

```
Class A {
    int y = 26;          → Instance/Non-static
    Static float z = 29.6f;   → static
    void display () {         → method
    boolean a = true;         → local
  X Static double b = 92.784; (Invalid)
    }
}
```

int x; → variable declaration

x = 72; → variable Initialisation

int z = 96; → variable declaration & Initialisation

Methods:

It is nothing but the block of statements or code, these statements are used to implement specific task or any operation.

The main purpose of a method to implement reusiblity of the logic (or) functionality.

Syntax of method:

```
Return type (or) Non-return type Method name (Parameters)
{
  // Block of Statement.
}
```

Types of methods:

1. Predefined methods.
2. User defined / custom defined methods.

predefined methods are in-built methods developed by Sun-micro Systems; they will deals with entire Java Software.

Custom defined methods need to be developed by either programmer or developer and they will deals with specific project or product.

These methods are sub classified into the following

1. Non static (or) Instance.
2. Static
3. Return type

4. Non-Return type
5. with parameters
6. with out parameters.

NS/I	Non-static Instance
NRT	Non-Return type
WOP	without parameters
WP	with parameters
RT	Return type
S	static

```
class Sample {
void m1 ( )
{
-> NS/I, NRT & WOP
}
Static void m2 ( )
{
-> S, NRT & WOP
}
void m3 (int x)
{
-> NS, NRT & WP
}
Static void m4 (char b)
{
-> S, NRT & WP
}
float m5 ( )
{
-> I, RT & WOP
}
Static boolean m6 ( )
{
-> S, RT & WOP
}
String m7 (float z)
{
-> NS, RT & WP
}
Static double m8 (boolean d)
{
-> S, RT & WP
}
}
```

OOPS Concept:

- → class
- → object
- → Inheritance
- → Data Hiding
- → Encapsulation
- → polymorphism
- → Abstraction
- → Dynamic Binding
- → Static Binding
- → Message Passing:

Class: It is the collection (or) group of Variables, Methods and constructors in a single unit (or) a Single block.

The main purpose of class is used to declare user defined data types.

It is an logical entity.

It contains only state (Name, Colour, type size etc but not behaviour (Action, functionality).

we can able to create the user defined class by using a pre defined keyword called class.

Syntax of a class:

```
class <classname>
{
Variable declaration & Initialisation
Method declaration & definition
}
```

Object:

It is an Instance of a class, Instance is nothing but allocating memory space for class properties (variables, methods & Constructors).

(or)

A Blueprint of a class nothing but object.

(or)

A real world entity nothing but object.

(or)

A variable of a class nothing but object.

It contains both State, and behaviour,

It is an physical entity.

Number of ways to create object:

By using 'new' key word.

By using factory method.

By using clone() method.

By using De-Serialization

By using getInstance()

By using newInstance()

Syntax to create object by using new keyword:

Default constructor

<class name> <object name> = new <class name>();

<class name> <object name>;

↳ object declaration

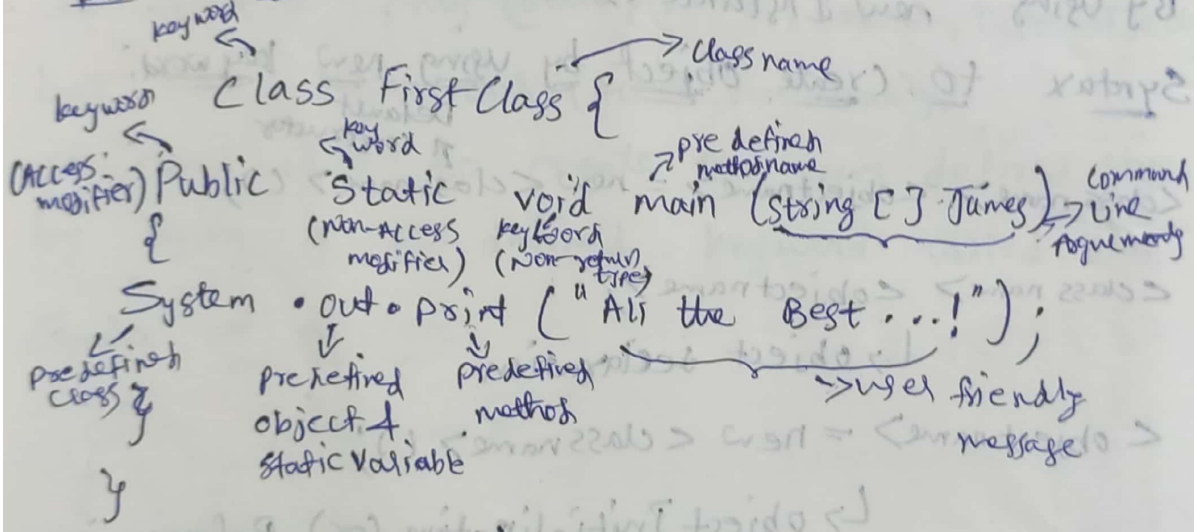
<object name> = new <class name>();

↳ object Initialization (or) Referencing

Structure of Java program:

```
Package details;
import statements;
class <class name>
{
variable declaration & Initialisation;
Method declaration & definition
Public Static void main (String [] args)
{
    // Block of statements.
}
}
```

Sample Program:



Differences between Static and Non-Static Properties

Difference between Static and Non-Static

Static	Non-Static
* The main purpose of Static property is to implement common operations (or) one time operations.	* The main purpose of Non-Static properties is to implement repeated (or) unique operations.
* For all Static properties we have to use Static keyword.	* Here we should not use Static keyword
* For Static properties memory space will be allocated only once that too on compile time.	* For Non-Static properties memory space will be allocated repeatedly depends on number of times of Object creation that too on Run time.
* If we want to invoke Static properties under either Static (or) Non-Static area we don't need object.	* If we want to invoke non-static properties under non-static area we don't need object, if we want to invoke Non-static properties under Static area we must need object
* Static properties can be access either through class name or directly or through object name.	Non-Static properties can be Access either directly or through object name but not through class name.
* Static method can be overload but cannot be override.	* Non-Static method can be both overload or override.

(1)
* Create java application where we have one class, it containg one non-static method like display, which is going to returen ~~written~~ as nothing and it does not have any formal parameter, ~~that~~ method has to print any user friendly message on the console while we invoke this method under main method.

(2)
* Create java application where we have one class, it contains one non-static method like display, it contains one parameter like string, which is going to print the value of String, this method is going to written nothing, then invoke this method under main method.

(3)
* Create java application where we have one class, it containg one non-static method like display, having parameter boolean, it has to print the value of boolean and it is going to writtens nothing then, invoke this method under main method.

(4)
* Create java application where we have one class, it contains one non-static method display, having parameter integer, it has to print the value of integer and which is going to writtens nothing then invoke this method under main method by Providing dynamic input.

⑤ create java application where we have one class, it contains non-static method M1, having parameters float, string, it has to print both float and string values and it has to written nothing, one static method M2 having parameters boolean and long, it has to print both values and it is going to written nothing, one non-static method M3 having parameters character and integer, it has to print these two values which is going to written nothing then invoke all these methods under main by providing dynamic inputs.

```
*
① class L
{
void display()
 {
    System.out.print("Hello world");
 }
Public static void main(String [] x)
{
 L  a=new L();
  a.display();
}
}
```

(4)

```
import java.util.Scanner;
class A
{
void display (int x)
{
   System.out.Print (x);
}
Public Static void main(String [] arg)
{
   A obj=new A();
Scanner sc=new Scanner (System.in);
System.out.print("Enter the int value");
int in=sc.nextInt();
obj.display (in);
}
}
```

```
import java.util.Scanner;
class X
{
Static Scanner obj = new Scanner (System.in);
 float m1 (String a)
 {
     System.out.print(a);
     System.out.print ("Enter float value");
     float fV = obj.nextFloat ();
     return fV;
 }
 boolean m2 (char b)
 {
     System.out.print (b);
     System.out.print ("Enter string value");
     String sV = obj.next();
     float res = m1 (sV);
     System.out.print (res);
     System.out.print ("Enter boolean value");
     boolean bV = obj.nextBoolean();
     return bV;
   }
public static void main (String [] xyz)
{
     X obj2 = new X ();
     System.out.print ("Enter char value");
     char cV = obj.next().charAt(0);
     boolean res2 = obj2.m2(cV);
     System.out.print (res2);
   }
}
```

Constructor

Constructor: It is also a type of method, It's one of the specially defined method and the main purpose of constructor is where we can able to create user defined object for respective class and we can provide user defined values for instance variables during object creation.

Rules about constructor:

* The constructor name must be same as class name.
* The constructor should not have any return type even non-return type void, If we give any return type for constructor then it will becomes a normal method.
* For constructor we should not use the key words like Static, final, Abstract and Synchronized.
* Constructor will invokes automatically while creating object.
* Constructor can be overload but cannot be override.

Types of Constructor:

1. Default constructor
2. Parameterized constructor.

Default Constructor:

* These constructor does not have any parameters.
* Compiler will creates it's own default constructor for every class.
* If we don't create explicitly.

Syntax:

```
class <class name>
{
    <class name>()
    {
    ==  // Block of statements.
    }
}
```

Parameterized constructor:

* These constructor will have respective formal parameters depends on requirements.
* By using these constructor we can able to provide user defined values for instance variables

Syntax:

```
class <class name>
{
    <class name>(formal parameters)
    {
    ==  // Block of statements
    }
}
```

Object parameterised Constructor:

* In these constructor we have object as a parameter, the main purpose of these constructor to copy the content from one object into another object that's why it is also known as copy constructor

Syntax:

```
class <class name>
{
    <class name> (<class name> <object name>)
    {
        // Block of statements
    }
}
```

This: It is a predefined keyword and also known as current class object.

whenever we have both instance variable and local variable both are same, JVM will get Ambiguity (confusion). to differentiate these variables, by default JVM will gives highest priority for local variable, if we want to highlight instance variable then we can access it either through object or by using this keyword.

we should not use this keyword under static area.

we can use this keyword at variable level, method level and constructor level.

Inheritance:

By using Inheritance we can implement Reuseability of the properties (variables & methods) from one class into another class by using a predefined keyword called extends.

Whatever class will gives its properties to another class then it is called either base class (or) Super class (or) Parent class.

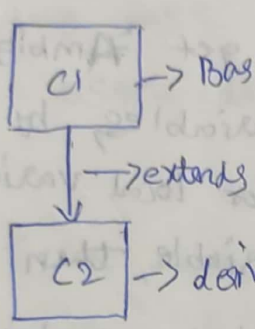
what ever class will acquires (or) Reuse the property from some other class then it is called either derived class (or) Sub class (or) child class.

Types of Inheritance:

1. Single - level - Inheritance.
2. Multi - level - Inheritance.
3. Hierarchial Inheritance.
4. Multiple - Inheritance.
5. Hybrid Inheritance.

Single level Inheritance:

Here we will acquire the properties from Single Base class to single derived class.

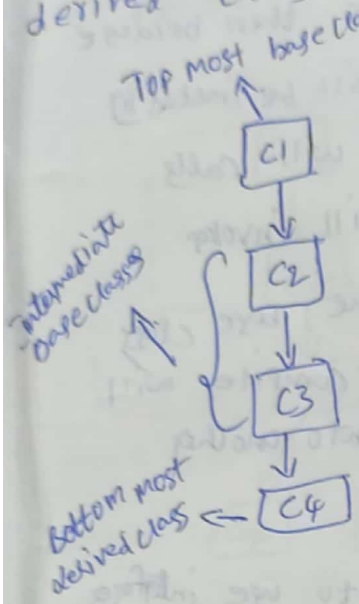


Syntax:

```
class <class name>
{
  // Block of Statements.
}
class<class name 2> extends <class name>
{
  // B. of. S
}
```

Multi level Inheritance:

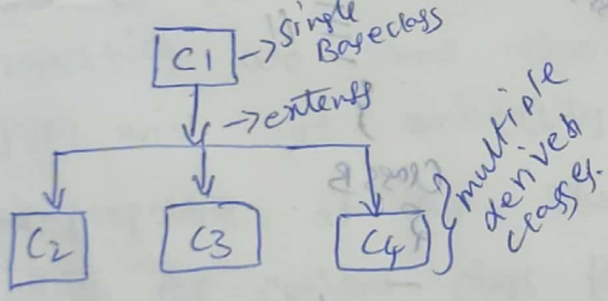
Here we will acquire the properties from Top most base class to intermediate base classes and then from Intermediate base classes to bottom most derived class.




```
class <class name>
{
== // B.O.S
}
class <class name 2> extends <class name>
{
== // B.O.S
}
class <class name 3> extends <class name 2>
{
== // B.O.S
}
```

Hierarchial Inheritance:

Here we will acquire the properties from single base class to multiple derived classes.




```
class <class name>
{
== // B.O.S
}
class <class name 2> extends <class name
{
== // B.O.S
}
class<class name 3> extends <class name
{
== // B.O.S
}
```

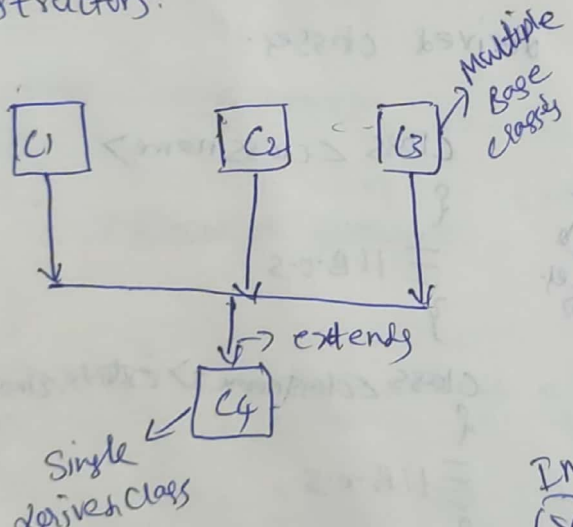
Multiple Inheritance:

Here the Single derived class will inherits more than one base class at a time, It is not possible in java through the classes because of following reasons.

Whenever any class will extends one more class then because of extends keyword Super keyword will be invoked by default, thats why base class constructor will calls first then derived class constructor will invoke

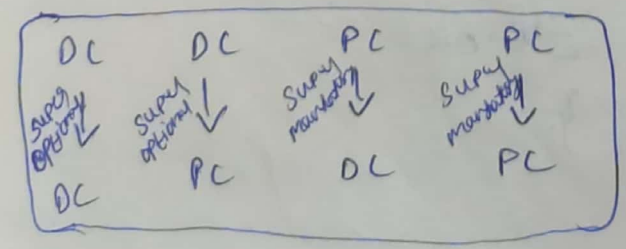
Whenever any class extends more than one base class Super keyword will be invoke and compiler will tries to invoke more than one constructor into another constructor but it is not possible.

To achieve multiple Inheritance we need to use interfaces Concept why because in Interfaces we don't have Constructors.



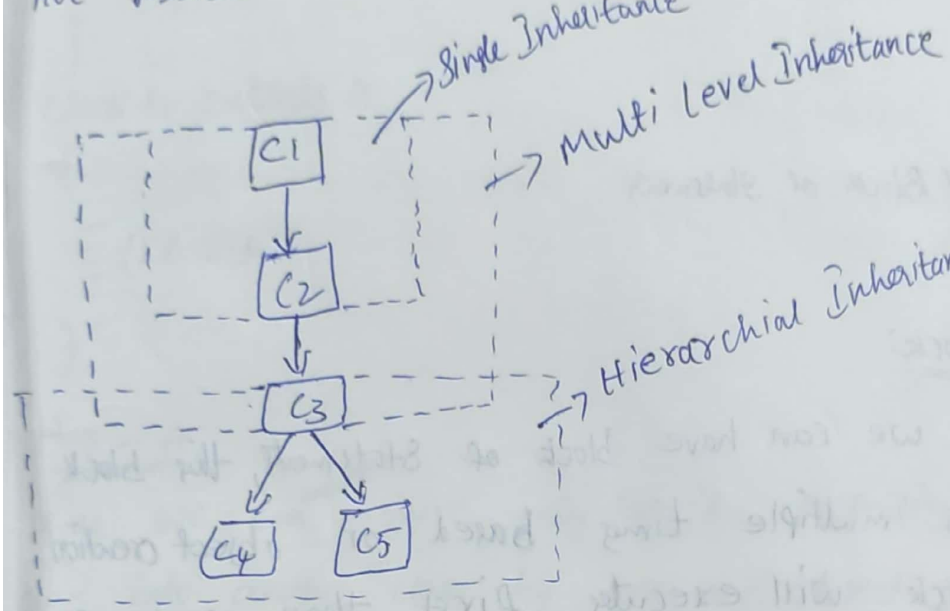

```
Class A {
    ≡ ≡ ≡
}
Class B
{
    ≡ ≡ ≡
}
Class C extends A,B
{
    ≡ ≡ ≡
}
```

Invalid



Hybrid Inheritance:

It is the combination of any two or more types of Inheritances, If in this combination any one type belongs to multiple Inheritance then entire that architecture is not possible.



Super: It is called either base class or Super class or parent class object.

whenever we are inheriting the properties from base class to derived class, if both base class and derived class properties are same then during execution point of time JVM will get ambiguity or confusion to differentiate the properties, at this situation JVM will gives highest priority for current class properties and if we want to highlight base class properties then we have to use Super keyword or we can able to access through base class object.

Super keyword plays a keyrole at three levels

1. At variable level.
2. At method level.
3. At constructor level.

Both This and Super are non-static we should not use under main

Static Block:

It contains particular block of statements, this block will contains more priority than main method.

So static block will executes first than main method

Syntax:

```
Static {
    // Block of statements
}
```

Instance Block:

Here also we can have block of statements, this block will executes multiple times based on object creation and this block will executes first than constructor.

Syntax:

```
{
    // Block of statements
}
```

Types of Relationships

we have three types

-> is - A
-> has - A
-> user - A

is - A

Here we will acquire all the properties from one class into another class by using extends keyword.

Here if we perform any changes in one class, those changes will difanetly will reflect in some other class (extended class)

Here irrespective of our requirements all properties of base class available for derived class.

```
class A
{
  // Block of statements
}
class B extends A
{
  // B.O.S
}
```

Has – A:

Here we can(able to) acquire only specific properties from one class into another class by creating object globally instead of using extends keyword.

Here if we are going to perform any changes in one class then there will be less impact on some other class.

Here we will give the access to all the methods of one class to reuse specific ~~methods~~ properties of some other class

```
class A
{
  // B.O.S
}
class B
{
A obj=new A();
  // B.O.S
}
```

User - A

Here we will do the same like as above but instead of creating object globally we will create object locally i.e. under only specific methods, so that we cannot share that specific object through out the class.

Here we will give the access to specific methods of one class to acquire specific properties of some other class.

```
class A
{
  ≡≡
}
class B
{
  void display()
  {
    A ai = new A();
    ≡≡
  }
  void m2()
  {
    ≡≡
  }
}
```

Data - Hiding: The process of Hiding confidential information from unauthorised user is called data Hiding.

we will implement this concept by using a pre-defined key word called private (access modifier)

If any property is private then we can able to access that property only within a class but we cannot access outside of the class.

we can use this private keyword at variable, method, constructor and inner class levels.

(inner class also known as Nested class)

eg:

```
class Bank {
    private int atm_pin = 4328;
    }
```

```
class Bank
{
  private int atm-pin = 4328;
      void SetAtmPin(int atm-pin)
      {
        this.Atmpin = atm-pin;
      }
        int getAtmpin()
        {
          return atm-pin;
        }
      }
```

Encapsulation: The process of tabing private fields (variable) binding with setter and getter methods

Set xxx()
get xxx()

By using setter methods we can reset (or) update the values for private fields and by using getter methods we can return the updated values.

Setter methods always have formal parameters and returns nothing, getter methods always has to return the respective values and should not have parameters

Note: To change (or) update the values for private fields we can use either setter injection or constructor injection

```
class Bank
{
 Private int atm_pin=4328;
    void setAtmpin(int atm_pin)
    this.atm_pin = atm_pin;
  }
  int getAtmpin()
  {
    return atm_pin;
  }
}
Class Test
{
  Public static void main (String [] args)
  {
    java.util.Scanner S1 = new Scanner java.util.Scanner (System.in);
    Bank obj = new Bank();
    obj.setAtmpin(S1.nextInt());
    System.out.print (obj.getAtmpin());
  }
}
```

Polymorphism: The process of represent ing one form into multiple forms is called polymorphism.

In this one form is called either original or Actual method, multiple forms are called either overloaded or overridden methods.

It is of two types.

1. Compile time polymorphism.
2. Run time polymorphism

Compile time polymorphism is also known as either overloading or static polymorphism or early binding.

Run time polymorphism is also known as either Dynamic polymorphism or over riding or late binding.

Method over loading:

Whenever two or more methods are having same method name, different signature (no.of parameters, type of parameters and order of parameters) and irrespective of return type It is going to be happened only within a class.

Syntax:

```
Class <class name>
{
  Return/non return type  method name (Parameters)
  {
    =
    =
    =
  }
  Non-Return type/ return type  Method name (Arguments)
  {
    =
    =
  }
}
```

Method over riding:

Whenever two or more methods are having same method name, same signature (parameters) and same return type with respect to extended classes or implementation classes or implemented classes.

The main purpose of method overriding is used to provide functionality for abstract methods with respect to either abstract classes or interfaces.

```
class <class name>
{
Return type/non return type methodname (Parameters)
{
  =
  =
}
}
class <class name> extends <class name>
{
  Return type/non Return type methodname (parameters)
  {
    =
    =
  }
}
```

Create java application where we have one class it contains 3 methods, which are having same method name with different parameters and return types then invoke all these methods under main by providing dynamic inputs.

Create java application where we have two classes with respect to is-> A relationship, which are having same method name, same parameter and same return type then invoke these two methods under main by providing dynamic inputs.

Create java application where we need to satisfy polymorphism (overloading and overriding).

Abstraction: The processes of hiding internal or background working mechanism and providing necessary information or services to the end user.

This Abstraction can be achieved by using a predefined keyword called Abstract

To implement the Abstraction mechanism we have two ways

1. By using Abstract classes
2. By using Interfaces

Types of classes:

1. Concrete class
2. Abstract class

Concrete class: This class will contains purely defined methods but it should not have atleast one abstract method.

Defined method will contains both declaration and definition

Defined method is also known as concrete method or implemented method.

Eg:

```
Void SendMoney(double amt) → method declaration.
{
 ----
 ----   // Block of statement / Business Logic
 ----
}
↓
method Definition.
```

Abstract class: This class will contains both defined and undefined methods.

Undefined method is also known as Abstract method or unimplemented method.

Abstract method will contains only declaration but not definition and this abstract method must be preceded by

Abstract key word and must ends with (;).

Eg: abstract void SendMoney (double amt); → method Declaration

Rules or Guidelines about Abstract classes:

* Abstract class must be preceded by Abstract keyword.
* In Abstract class we can have either only defined methods (or) only Abstract methods (or) combination of both defined & Abstract methods.
* we will inherit the properties from Abstract class to Abstract class (or) Abstract to concrete class (or) concrete to Abstract by using extends keyword.
* whenever we are inheriting the properties from Abstract to concrete class then each and every Abstract method must be override into it's extended classes (or) implementation classes.
* By using Abstract classes we can achieve 0 to 100% range of Abstraction.
* For Abstract classes we cannot create object directly but we can do indirectly either by using dynamic binding principle (or) Upcasting.
* Every Abstract method of Abstract class must be preceded by Abstract key word.

Note: Abstract method should not be Static why because we cannot override Static method.

Syntax:

```
abstract class <class name>
{
    variable declaration (or) Initialisation;
    method declaration (or) definition
}
```

Create Java Application where we have one abstract class, it containg two abstract methods and one defined method with parameters and return type then inherit this abstract class into concrete class, Provide implementation for abstract methods by using overriding concept and then invoke all the methods under main method by providing dynamic inputs.

Interfaces: Interface is also a type of class, it is the collection of public, static, final variables and public abstract methods.

Rules (or) guidlings about interfaces:

* Interface will be created by using a predefined keyword called Interface.
* Interface variables can be access anywhere why because they are by default public.
* For interface variables memory space will be allocated only once why because they are by default static.
* Interface variable values cannot be modified why because they are by default final.
* For Abstract methods we need not to use Abstract keyword manually because Abstract methods of interfaces are by default both public and Abstract.
* we will inherit the properties from interface to interface by using extends keyword, interface to class

(concrete (or) Abstract) implements keyword.

* Whenever we are inheriting the properties from interface to concrete class then each and every Abstract method must be override into it's implementation classes and every overriden method must be public because interface Abstract methods are by default public.
* In interfaces we cannot have constructors.
* By using interfaces we can achieve 100% abstraction and multiple inheritance.
* Upto java 7 version in interfaces we have only Abstract methods, in java 8 version we can have normal methods also but these methods must be preceded by either static or default keyword.
* Interfaces cannot be instatiated (Object creation) directly but we can do indirectly by using dynamic binding (or) upcasting.
* There is no super class for all the interfaces.

Syntax:

```
interface <Interface name>
{
  variable declaration & Initialisation;
  method declaration (upto JDK 7)
  method declaration or definition (In Java 8 version)
}
```

Create Java application where we have two interfaces with respective to individual Abstract methods then inherit these two interfaces into one more interface there we have one Abstract method and one defined method then inherit these interface into concrete class, provide functionality for Abstract methods and then invoke all these methods under main by providing dynamic inputs.

Inner classes (or) Nested classes:-

Here we have four types of Inner classes.

1. Member Inner class
2. Local Inner class
3. Static Inner class
4. Annonymous Inner class.

1. **Member Inner class**: Here we will create class inside Some other class (or) Another class

The main purpose of this Inner class is where we can access the private properties from one class into another class.

Syntax:

```
class <classname>
{
  == // B.O.S
  class <classname 2>
  {
    == // B.O.S
  }
}
```

Create java application where we have one class it contains one private variable and one method with parameter and return type, Inside this class we have two Inner Classes, It contains one method with Parameter and return types Individually. these methods has to access the private variable and then we need to invoke all these methods under main by providing dynamic inputs.

Local Inner class: Here we will create class inside method of outer class (or) another class.

Syntax:

```
class <class name>
{
  // statement
Return/non Return type method name (parameters)
  {
  // statement
  class <class name2>
  {
  // statement.
  }
  }
}
```

Create java Application where we have one class it contains one method with parameter and return type, inside this method we have one inner class containing two methods with parameters and return types then invoke all these methods by providing dynamic inputs.

Create java application where we have one class, it contains one method with parameter and return type, inside this class we have another class, it contains one more method with parameter and return type, inside this method we have two inner classes with respect to individual methods with parameter and return types then invoke all these methods by providing dynamic inputs.

Static Inner class: Here we will create class which is of Static inside some other class.

Syntax:

```
class <class name>
{
  ==
  Static class <class name>
  {
    ==
  }
}
```

Create java Application where we have one class, it contains one method wit parameters and return type, inside this class we have another class which is of static, here also we have one method with parameter and return type, then invoke these two methods under main by providing dynamic inputs.

Annonymous Inner class: Here we have one class, it doesnot have any class name, where it can extends one class or it can implements only one interface at a time

Syntax:

```
class <class name>
{
return type methodname (Parameters)
{
 ---
}
Public static void main ( )
{
<class name>  <objectname> = new <classname> ()
{
  return type method name (parameters)
  {
  ---
  }
  };
}
}
```

Nested Interface: Here we will create interface inside another interface.

Syntax:

```
interface <Interface name>
{
   ==
   interface <Interface name>
   {
      ==
   }
}
```

Create a java Application where we have one interface it containg one abstract method and one defined method, inside this interface we have two interfaces with respect to individual abstract methods then provide implementation for all these interfaces into one concrete class then invoke all these methods under main by providing dynamic input

Packages

Sun Micro Systems has provided a in built facilities like generic API [Application programming Interface]

* API is collection of classes, Interfaces and subpackages
* These multiple classes and Interfaces Internally contains variables, methods and constructors.
* By using this API we can able to develop different types of Applications.
* The main purpose of packages is used to create custom defined folders in respective applications, we can maintain different types of files in different folders and it will takes less amount of time to access respective files from respective folders.
* By using Inheritance we can able to access the properties within a file but not outside of the file but by using packages either we can access within a file or out of the file also.
* packages are mainly classified into two types.

1. Pre defined (or) [In built packages).
2. Custom defined (or) user defined packages.

* By using pre defined packages we can develop different types of applications with respect to java software and to satisfies universal requirements.
* By using custom defined packages we can develop different types of applications with respect to particular product (or) particular service and to satisfies particular requirements.
* Pre defined packages are classified into following types

1. JSE packages. (Java Standard edition)
2. JEE packages. (Java enterprise edition)
3. JME packages. (Java mobile/micro edition)

C- Third party vendor packages

* By using JSE packages we can develop desktop or stand alone applications.
* By using JEE packages we can develop distributed applications.
* By using JME packages we can develop mobile applications
* By using Third party vendor packages we can develop different types of Applications. by using particular software which is developed by different vendor.

JSE Packages

-> java.lang.*:-

By using this package we will get following services

1. It will provides language functionalities.
2. It will provides garbage collection facility, String handling functions, exception handling techniques, multithreading facility etc.,
3. It is a default package in java and we need not to import this package explicitly or manually.
4. The super class for all the class is java.lang.Object.

-> java.awt.*:-

By using this package we can able to create window based GUI components but not functionality.

-> java.awt.event.*

* By using this package we can able to provide functionality or behaviours to the GUI components.
* By utilising these two packages we can able to develop window based GUI applications.

→ java.util.*

By using this package we can able to develop collection framework applications and we can provide dynamic inputs.

→ java.net.*

By using this package we can develop network based applications like client to server, server to database etc.,

→ java.io.*

This package used to develop file based applications

→ java.sql.*

By using this package we can deal with JDBC (Java DataBase connectivity) Applications.

Steps (or) Guidelines to create Custom defined (or) User defined Packages:-

* Initially we have to create the user defined package by using a predefined key word called package.
* with respect to the package whatever class or interface which we will create, they must be public
* whatever properties which we will create, with respect to the class, they also must be public.
* Interface properties are by default public.
* whatever class or Interface which we will create, their names must be file names.

How to Access the properties from packages:

* By using import key word
* By using fully qualified name Approach.

import key word:

case-1:

Here we can access particular properties from particular class or Interface from particular package into either seperate package (or) seperate file

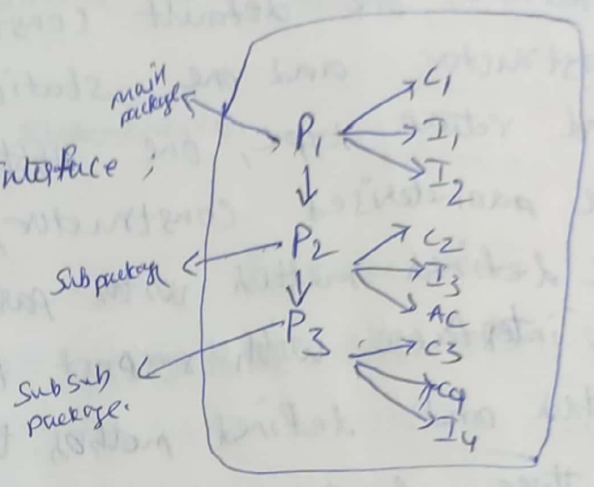
class(1)

import packagename.classname/Interface;

import P1.P2.P3.C4;

import P1.P2.AC;

import P1.I1;



(case(2))

Here we can access all the properties from all the classes and interfaces from particular package into seperate package but not seperate file

import package name.*;

import P1.P2.P3.*;

import P1.P2.*;

import P1.*;

Fully qualified name approach:

Here we can able to access the properties from particular class or particular interface from particular package into either seperate package (or) seperate file without using import keyword

P1.P2.P3.C3 obj = new P1.P2.P3.C3();

* Create java Application where we have one package it contains one concrete class with variable, one Parameterised constructor and one non static method with parameter and return type then access all these properties into the class of seperate package by using import key word.

* create java application where we have one package, it contains one concrete class with one non static variable, one default constructor, one Parameterised constructor and one static method with parameter and return type, one Abstract class it contains one parameterised constructor, one abstract method any one defined method with parameters and return type, two interfaces with respect to individual abstract method and defined methods. then we will do Access all these features into the class of seperate package, Provide functionality for Abstract methods and then invoke all these properties under main by providing dynamic inputs.

Create java application where we have one package, it contains one concrete class with one parameterised constructor and one method with parameter and return type, one abstract class it contains one parameterised constructor, one abstract method and one defined method with parameter and return type, one interface it contains one defined and one abstract method with parameter and return types, with respect to this package we have one sub package it contains two interfaces with respect to individual abstract and defined methods with Parameters and return types, with respect to this package we have one sub sub package it contains one

concrete class with parameterised constructor and defined method with parameter and return type, two interfaces with respect to individual abstract and defined methods with parameters and return types then access all these features into seperate class of seperate package like main package and subpackage features by using import keyword and subsub package features by using fully qualified name approach.

~~Run~~ ~~javac -d . filename.java~~

~~Compile~~

compile - javac -d . filename.java

Run - java packagename.classname

* Modifiers *

It is of two types. 1) Access Modifiers
2) Non-Access Modifiers

1) Access Modifiers:- These Modifiers will gives us Scope (or) ranges of the properties like upto how far we can access and where we can't access. The following keywords are the access modifiers.

1. private.
2. Default.
3. protected.
4. public

1) private: The Scope of the modifier is we can able to access the properties only with in a class but not outside of the class. Even if we do (is-A relation)

=> we can use this keyword at variable, method and constructor and inner class levels.

=> It is also known as "native access modifier".

2) Default: The Scope of this modifier is we can able to access the property with in a package but not outside of the package.

=> It will be there by default at variable, method, constructor, class & Interface but we should not use this keyword explicitly (or) manually.

=> we can use this keyword explicitly only at defining method of interface (Java-8 version).

=> It is also known as "Java access modifier".

protected: The scope of these modifier is we can able to Access the properties outside of the package but only through sub class.
we can use this keyword at variable, method and constructor levels.
It is also known as inherited Access modifier.

Public:
The scope of this modifier is we can able to access the properties anywhere either in/d package or outside of the package.
We can use this keyword at variable, method, constructor, class and interface levels.
It is also known as "universal Access modifiers".

Create a java application where we need to satisfy the scope of all access modifiers.

```
// A.java
Package P1;
Public class A
{ private int a=10;
  float b=1.67f;
  protect boolean c=true;
  public double d=784;
  public void m1()
  {
    S.o.p("method");
  }
}
```

```
//B.java
package P1;
public class B extends A
{
  public void m1()
  { S.o.p(d);
  }
}
```

```
// C.java
package P2;
import P1.*;
```

```
//c.java
package p2;
import p1.*;
public class c extends B
{
  public void m1()
  {
    S.o.p(b+" "+c+" "+d);
    S.o.p(new c().c);
  }
}
class test
{
  p s v m ( )
  {
    A obj = new A ();
    obj.m1();
  B obj = new B();
    obj2.m2();
  (obj3 = new c();
    obj3.m3();
  }
}
```

Non-Access Modifiers:- The main purpose of these modifiers allows us to apply conditions on particular applications then based on those conditions the program behaviours will depends.

Following are the non-access modifiers.

1. static
2. final
3. Abstract
4. Synchronised
5. transient
6. volatile
7. strictfp.

1) Static:- The main purpose of static keyword is to implement either common operation (or) one time operation.

⇒ It will gives us efficient memory management.

⇒ We can use static keyword at variable, method, inner class level and block levels.

⇒ Static method can be overload but can not be override.

⇒ for these properties memory space will be allocated only once at the time of compilation and we need not to create object to invoke these properties.

2) Final:- The main purpose of final keyword to make the things as constant.

⇒ we can use these key word at variable, method and class levels.

⇒ If we use these key word at variable level then we can't change it's value.

⇒ If we use these keyword at method level then that method can be overload but can't be override.

⇒ If we use these keyword at class level then that class can't be inherited.

```
class X
{
final int a=10;
void display()
{
  a=a+20;   (X)
  int b=a+40;
  S.o.p(a);
}
}
```

```
class A
{
final void m1(int a)
{
  S.o.p(a);
}
void m1(String a)
{
  S.o.p(a);
}
}
class B extends A
{
  void m1(int a)
  {
    S.o.p(a);
  }
}
```

(X)

```
final class X
{
  ==
}
class y extends X
{
  ==
}
```

3) Abstract:- The main purpose of abstract keyword is to implement abstraction.

=> To achieve abstraction we have 2 ways.

1. By using abstract class.
2. Interfaces.

=> we can use this keyword at variable method level and class levels.

4) Synchronised: The main purpose of Synchronise keyword is we to implement Synchronisation concepts.

=> Actually the main purpose of multithreading will provide parallel flow of execution.

=> If multiple threads are running parallel on the same resource then we will get inconsistant results

⇒ To get consistant results we need to apply synchronisation concept.

⇒ synchronisation is the process of allowing thread by thread instead of parallel (or) concurrent flow of threads. so that by implementing locking and unlocking mechanism. each and every thread result are going to be added each other but not replacing each other.

⇒ we can use synchronise keyword at method and block level.

5) transient: The main purpose of transient keyword is used to restrict not to be serializable of any fields (variables) binding with setter and getter methods.

Serialization: It is the process of transfering n no. of bytes of data in terms of one object into external file like .txt file.

Deserialization: It is the process of reading n no. of bytes of data in terms of one object from external file java application.

* we can use transient keyword at only variable level.

6) volatile: The main purpose of volatile keyword is to get consistent results whenever multiple threads are running parallely. * If the single variable accessed by multiple threads at atime then we can use volatile.

* It is an alternate option for synchronised keyword
* we can use this keyword only at variable level.

7) strictfp :- The main purpose of this keyword is used to maintain same (common) floating point range in each and every os.

* we can use this keyword at class, interface and method level.

Exception Handling

Actually in every program we might get two types of errors. 1. Compile time error.
2. Run time error.

* Compile time errors will arise whenever the user misuses the syntax of the Source code.
* Runtime errors will occurs on runtime whenever the user will gives invalid inputs.

Exception: It is error related statement (or) block of code (or) the statements which will contains invalid inputs, because of this the entire program flow of execution will be interrupted (or) disturbed.

* whenever the exception will occurs then the next line of code will never execute until unless by handling this exception.

* Types of exceptions:-

Exceptions are mainly classified into 2 types.
1. predefined exceptions.
2. User defined exceptions.

* pre defined exceptions are in built exceptions which are already developed by Sun micro systems. They are dealing with entire java software users to satisfies universal errors like division by zero error, invalid bounds of the error, invalid numbers format etc.

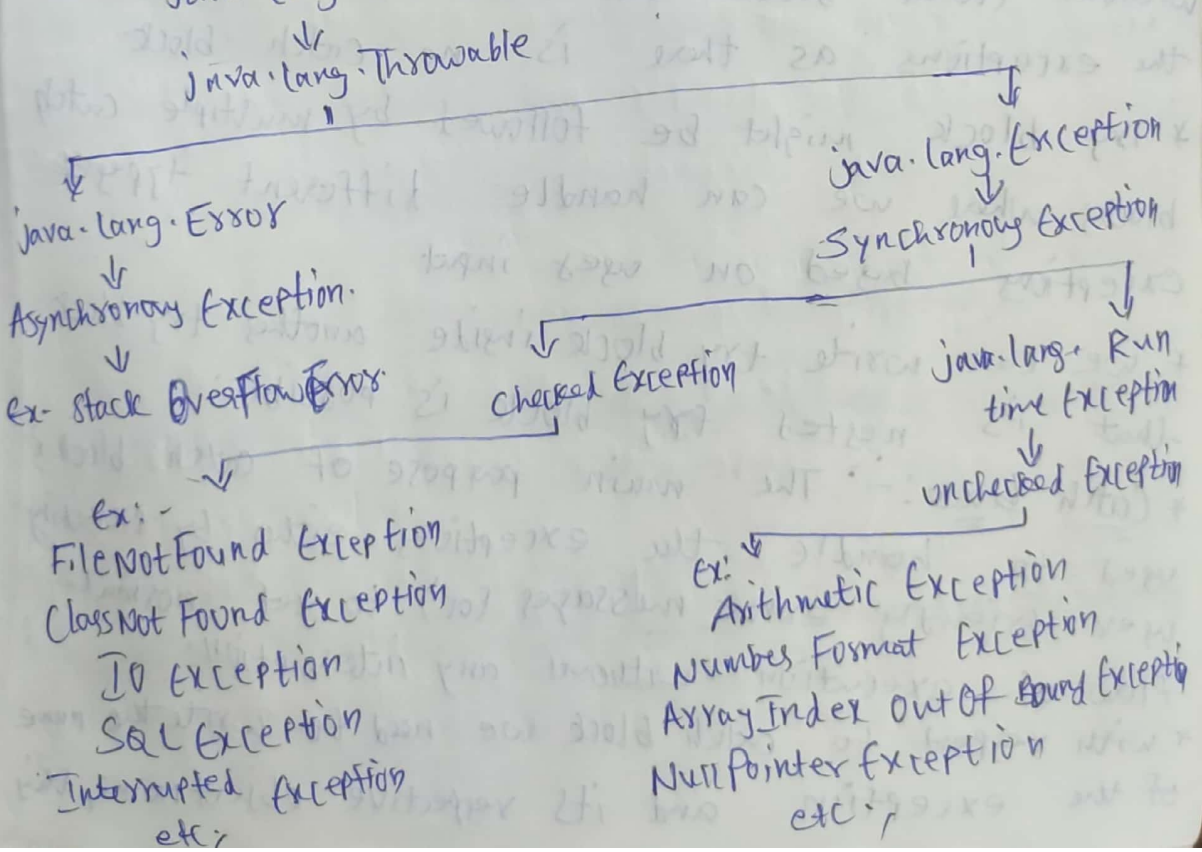
* predefined exceptions are sub classified into 2 types. 1. Asynchronous exceptions.
2. Synchronous exceptions.

* Asynchronous exceptions are dealing with hardware issues like input devices not working (or) hard disk failure etc; improper functioning of the software
* Synchronous exceptions are dealing with software code issues.
* Synchronous exceptions are again sub classified into 2 types. 1. checked exceptions.
2. unchecked exceptions

1. checked exceptions: These exceptions will occur at compile time error.
* These exceptions must need to be handle.

2. unchecked exceptions: These exceptions will arises at runtime.
* For few unchecked exceptions we will provide user friendly error messages and it is optional to handle few unchecked exceptions.

Exception handling:



* The Superclass for all classes is java.lang.Object.
* The Super class for all exceptions is java.lang.Throwable
* The Super class for all Asynchronous exceptions: java.lang.error.
* The Super class for all Synchronous exceptions java.lang.exception

Exception Handling:- By using this we can able to handle the exceptions either by providing user friendly error messages (or) by making a normal flow of execution without any interruptions

* To implement this concept we have following keywords try, catch, finally, throws & throw.

* Try block:- Here we need to write error related code (or) we need to maintain the statements which will raises the exceptions.

* Try block must be followed by catch block if we want to handle the exceptions.

* Try block directly followed by finally block without catch block but here we cannot handle the exceptions as there is no catch block.

* Try block might be followed by multiple catch blocks where we can handle different types of exceptions based on user input

* we can write try block inside another try block that is nested try block is possible.

* Catch block:- The main purpose of catch blocks used to handle the exceptions either by providing user friendly error messages (or) to get normal flow of execution without any interruption.

* with respect to catch block we need to write the name of the exception and its respective object name, this

object will be referenced automatically by JVM.

* with respect to catch block we can write try and catch block.

* Finally block:- The main purpose of finally block is used to print particular block of statements definately irrespective of exceptions handled (or) not

* with respective finally block we can write the statements which are used to close or release the resource objects.

* It is not mandatory to implement finally block in each and every ~~scenario~~ Scenario.

* we can write try and catch block inside finally block.

Syntax to implement try, catch & finally:

```
class <class name>
{
  == // statements
  RT / NRT method name (parameters)
  {
    == // statements
  try
  {
    == // error related code
  }
catch (<Name of the exception> <objectname>)
{
  == // Statements to handle exceptions (or) to provide user friendly
                                    error messages.
}
finally
{
  == // statements will execute definately irrespective of exceptions
                                    handled (or) not
}}
}
```

* Number of ways to handle unknown exception

1. By using exception class:-

If we use this technique it will give us two informations like 1. Name of the exception.
2. Nature of the message.

2. By using printstract Trace() method:

* It is predefined non-static method available in exception classes.
* This method is going to return nothing.
* If we use this method then it will give us name of the exception, nature of the message and the line number where the exception occurs.

3. By using getMessage():-

* It is a predefined non-static method available in exception classes, whose return type is String
* If we use this method then it will give us only nature of the message.

Throws: It is a predefined keyword, by using this we can able to handle only checked exceptions but not unchecked exceptions without using try and catch blocks.

* By using this keyword we cannot provide any user friendly error message to the user.
* we have to use this keyword at method heading

Syntax:

```
RT/NRT methodname (parameters) throws <Name of Exception>
{
  // Statements
}
```

Steps (or) Guidelines to create custom defined exceptions (or) user defined exceptions :-

* Initially we have to create user defined package.
* with respect to the package we need to create respective class as exception name, this class must extends either exception class (or) run time exception class and it must be public.
* with respect to the exception class we need to create parameterized constructor, parameter like String and String represents nature of the message and it must be public.
* we need to invoke parameterized constructor of either exception class (or) runtime exception class by using super keyword.
* whatever exception class we are creating, that class name must be file name.

Throw:

* It is also predefined keyword, the main purpose of this keyword to raise custom defined exception.
* we have to use this keyword inside method body.
* whenever we use throw keyword inside method body to raise custom defined exceptions then we must need to use throws keyword at method heading.

Syntax:

```
RT (NRT methodname (parameters) throws <name of exception>
{
  ==(statements
<name of exception> <obj.name> = new <name of exception>(string);
                                      nature of message
  throw <obj.name>;
}
```

MultiThreading

The main purpose of multitasking is used to implem two (or) more operations at a time.

Multitasking is of 2 types.

1. process based multitasking.
2. Thread based multitasking.

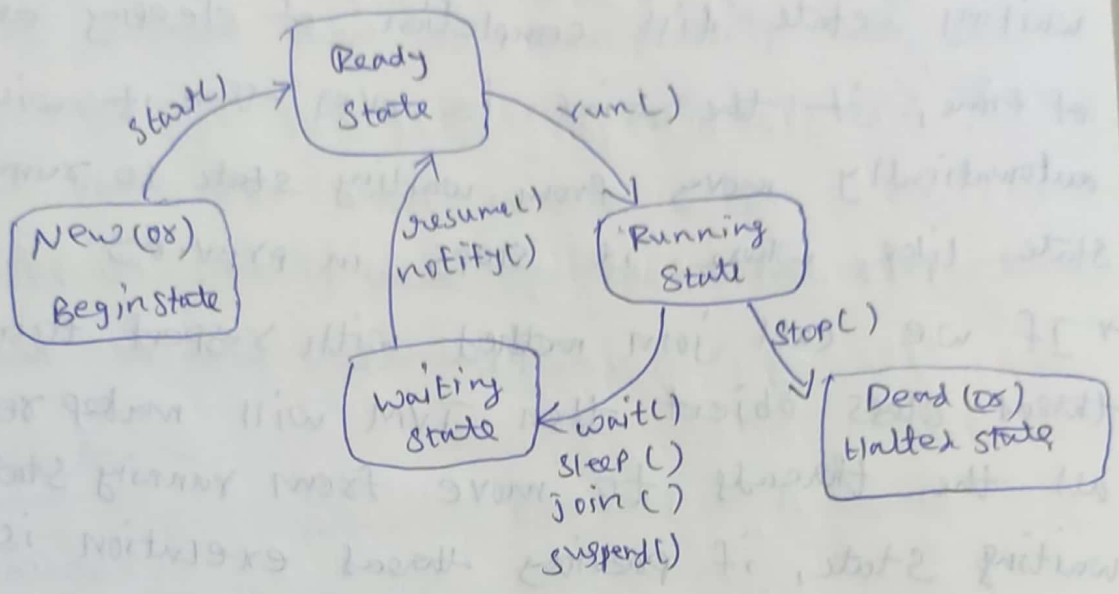
* Thread based multitasking is nothing but multithread
* The main purpose of multithreading is will provides parallel (or) concurrent flow of execution of multiple threads at a time.

* Differences between process based & thread based application.

process based application	Thread based application
1. under process based it contains single flow of control.	1. under thread based it contains multiple flow of control.
2. Here for each and every process a seperate memory space will be created.	2. Here the single address space willbe divided into sub memory spaces for respective threads.
3. It will takes more memory consumption.	3. It will takes less memory consumption.
4. It can be considered as heavy weight components.	4. It can be considered as less weight components.
5. Ex: C, COBOL, C++ are considered as process based technologies	5. Ex: Java, .NET, python are considered as thread based technologies.

Thread: It is an light weight component (or) It is a flow of control (or) It is one of the process and we can say it is a predefined class present in java.lang.package.

* Life cycle of Thread:



Explaination about thread life cycle:-

* whenever we create thread class object 1st time then that thread will be available in new state, if we invoke start method with respect to thread class objects then the thread will moves from new state to ready state, in this state the CPU will allocates a particular memory space and then after getting CPU permission the thread will automatically invoke run method, by invoking run method the thread will moves from ready to running state.

* If we want to make any currently running thread to move from running state to waiting state we can apply either (wait(), sleep(), join(), suspend method).

* After invoking wait method running state to waiting state, the thread will be in waiting state only until we call notify method.
* If we call sleep method by providing parameter value as long millisec, the thread will be in waiting state till completion of sleeping amount of time, if the time is over then it will be automatically moves from waiting state to running state like where it stops in previous.
* If we call join method with respect to any thread class object then JVM will make remaining all the threads to move from running state to waiting state, if previous thread execution is over then the remaining threads will automatically moves from waiting state to running state.
* If we call suspend method then that particular thread will be suspended from its execution, suddenly then it will moves from running state to waiting state, this thread will be in waiting state only until we call resume method.
* If we call stop method then that particular thread will moves from running state to dead (or) halted state and stop its execution permanently.

Profile of thread class:

variables: → public static final int MAX_PRIORITY.
→ public static final int MIN_PRIORITY.
→ public static final int NORM_PRIORITY.

* These variables are considered as priority modifiers by using these variables we can able to provide priority values for the threads then the based on priority values the Threads will execute one after another.

Note: The default value of max priority is 2, the default value of min priority is 1 and the default value for Norm priority is 5.

* The default priority value for each and every thread is Norm priority that is 5.

Constructors:-

→ Thread()
→ Thread (String)
→ Thread (Runnable)
→ Thread (Runnable, String)

1. Thread(): - By using this constructor we can able to create thread class object without providing any name.

Eg: Thread t_1 = new Thread();

2. Thread (String): - Here we can create thread class object by providing respective name.

Eg: Thread t_1 = new Thread ("Sai");

3. Thread (Runnable): - Here we will create thread class object indirectly with respect to runnable interface.

Eg: class A implements Runnable
{
≡
A obj=new A();
Thread t1=new Thread(obj);
}

* Here we are not going to provide any name to the thread class.

4. Thread(Runnable, String);- Here we will do the same like as above but here we can provide name also.

Eg:- class A implements Runnable
{
≡
A obj=new A();
Thread t1=new Thread(obj, "Vagtali");
}

Methods:-

→ public void setName(String)

→ public String getName()

*The main purpose of setName method is used to provide user friendly name to the thread class object.

*The main purpose of getName method is used to return respective thread class object name.

Number of ways to create thread class:

1. By using new keyword.
2. By using factory method.
3. By using custom defined thread with respect to thread class.
4. By using custom defined thread with respect to runnable interface.

By using new keyword:

Eg: Thread t1 = new Thread();

By using factory method:

factory method: A factory method is one which is static and whose return type is class name

Eg:

```
class A
{
    static A m1()
    {
        return new A();
    }
    PSVM () {
    A obj = A.m1();
    }
}
```

(FM)

In thread class we have one factory method like ~~current thread~~ currentThread.

Eg:

```
public class Thread
{
public static Thread currentThread()
{
    Thread t1 = Thread.currentThread();
    =
    ;
}
```

By using custom defined thread with respect to thread class

Eg:

```
            [custom defined thread]
Class A extends Thread -> [user defined thread]
{
   ⋮
   A a1 = new A();
}
```

By using custom defined thread with respect to runnable interface:-

Eg:

```
Class A implements Runnable
{
   ⋮
   A a1 = new A();
   Thread t1 = new Thread (a1);
}
```

```
Thread t1=new Thread ();
Thread t2=new Thread ();
Thread t3 = Thread. CurrentThread();
Thread t4 = Thread. CurrentThread();
```

Public void Run method:

* It is an abstract method available in Runnable interface, from this interface this Run method overridden into thread class.

* Whenever we create custom defined thread class then it is highly recommended to implement the logic by overriding run method with respect to custom defined thread class.

* This method will be invoked automatically if we call start method.

Public Void Start method : Start ();

* It is a predefined non static method available in thread class, if we call this method throw thread class object then it will execute Run method automatically after calling this method the thread will be allocated by particulas memory space and then it will makes the thread to move from new (or) born state to Ready state.

note: if we call Run method directly without using start method then we can able to execute normal functionality whatever we implemented under Run method, then this program will not works like threading program and this application will not get the multithreading services like synchronization, concurrency and inter thread Communication

* if we call Run method throw start method then this application will works like multithreading and we will get all multithreading services.

Public Static Void Sleep :-

* It is a predefined Static method available in thread class, we need to invoke this method throw thread class name.

* This method will takes long millisec as a parameter, if we call Sleep method with respect to any thread then the thread will moves from running state to waiting state, the thread will take sleep until completion of sleeping amount of time, if the time is over then this thread will automatically moves from waiting state to running state where it stops in previous

* This method will raise a predefined checked exception like 'java.lang.InterruptedException' So we must need to call this method with respect to try and catch to handle the exception.

Note: 1sec = 1000 millisec.

Public void join():

* The main purpose of this method is used to give more priority for currently working/Running Thread and at this point of time Remaining all the Threads will moves from Running state to waiting state.
* These Threads will be under waiting state only until completion of currently Running Threads.
* If it is over then it will makes the remaining Threads to move from waiting state to Running state Automatically
* This method also will rise a predefined checked exception like - interrupted exception, so we must need to invoke join method with in Try & catch blocks.
* we need to call join method after invoking start method.

Public void Suspend():

This method will makes the currently working Thread Suspended from its execution and it will moves from Running state to waiting state.

Public void Resume():

This method will makes the suspended thread to move from waiting state to Running state and restart the process.

Public

* The status thread of Dead stat

* I wai sta

Sy

x]

Scanned with CamScanner

Public boolean isAlive():-

* The main purpose of this method is will give the current status of working Thread like whether particular thread is under control of CPU (or) out of control of CPU, if the thread is in either new state (or) dead state then this method will gives current status as false (out of control of CPU).

* If the Thread in either Ready, Running and waiting state, this method will gives current status as True (under control of CPU).

Syncronization:

* It is the process of whenever multiple threads are running parallely on the same resource then the thread results are going to be replacing eachother thread results
* Hence we will get inconsistent ~~result~~ (garbage) Results
* By using synchronization concept we can avoid parllel flow of execution and it will allows thread by thread transaction.
* we can implement synchronization concept by using a predefined keyword 'synchronised'.
* we can use synchronised keyword at both method level and block level.
* If all the lines of the code of particular method need synchronization then we need to apply synchronised keyword at method level.
* If particular lines of code dealing with synchronization of particular method then we need to apply 'synchronised' keyword at block level.
* Synchronization concept will works by locking and unlocking mechanism.

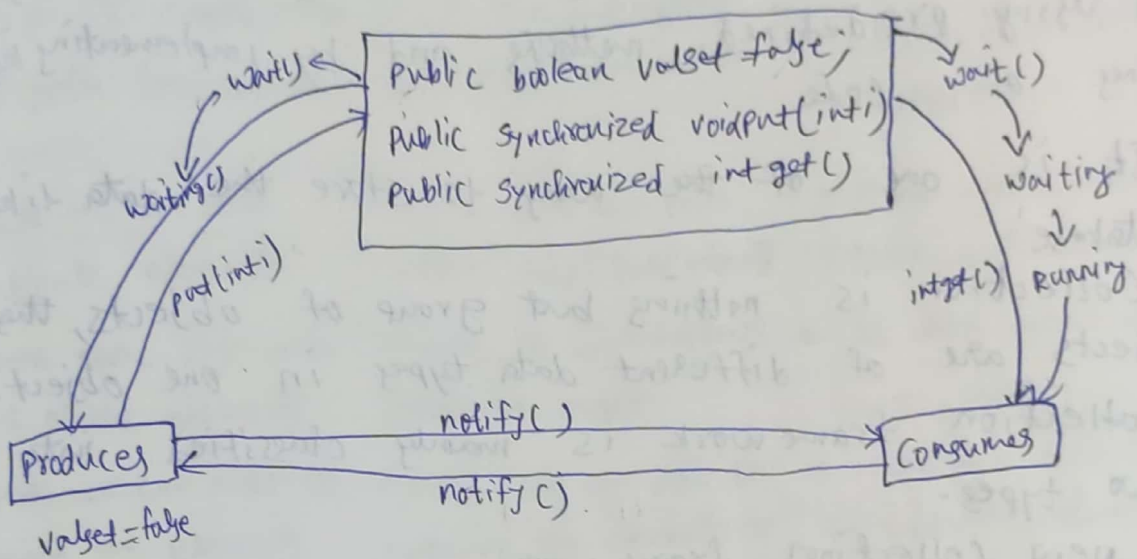
Eg:

* Suppose for example we have one balance variable whose initial value is '3000'.
* we need to deposit 5000 rupees into balance variable using two threads transactions.
* By using thread t_1 we need to deposit 2000 rupees and by using thread t_2 we need to deposit 3000 rupees
* If there is no synchronization, because of parallel flow of execution the thread results will be replaces each other so that the value of balance becomes either 2000 (or) 3000 (or) '3000'
* After applying synchronization, JVM will initially takes balance as '3000' and give it for thread t_1 to deposit 2000, After giving like this JVM will lock balance variable resource, After completion of the t_1 execution JVM will unlocks the resource and then it will add thread t_1 result to existing balance and the current balance is 2000, JVM will give this 2000 to thread t_2 to deposit 3000, After giving JVM will locks this resource again, After completion of thread t_2 JVM will unlocks again balance Resource and it will adds thread t_2 result to the existing balance and the current balance is 5000 rupees; After completion of these two threads execution JVM will hand over these two threads to the garbage collector,

Hence we will get exact results after implementing synchronization concept.

Inter Thread communication:

* Whenever multiple threads are going to be communicating each other by exchanging their input and output values with consistent communication.
* In this process we need to apply synchronization, wait method and notify method.
* If we invoke wait method the thread will move from running state to waiting state.
* If we call notify method the thread will move from waiting state to running state.



Thread Scheduler:

* The main purpose of this will take care about entire flow of threads execution like which thread has to execute first and then next and then last.
* It will works on two levels.

1. preemptive scheduling
2. Time slicing.

* By using preemptive scheduling we can set priority value for respective threads. then based on priority values thread will execute one-by-one we can provide priority values by using the method like setPriority(int)
* with repect to time slicing module the thread scheduler will allocates particular seconds of time for particular threads then the threads are going to run randomly (or) concurrently

like one thread few seconds of time then it will wa
for other threads then again it will moves for som
other thread randomly.

* The default priority of each and every thread is
NORM Priority and value is 5.

Collection Frame work

*By using Collection Framework we can able to store different types of values in single object, we ca
able to perform searching and sorting operations easi
by using predefined methods and by implementing les
lines of code.

* It is one of the way to store the data like
database.
* Collection is nothing but group of objects, these
objects are of different data types in one object.
* Collection framework is mainly classified into
two types.
1. New collection frame work.
2. Legacy collection frame work.

New collection frame work:

New collection frame work is sub classified into two
types.
1. 1 Dimensional new collection frame work.
2. 2 Dimensional new collection frame work.

one dimensional new collection frame work contains
different types of interfaces and classes; by using
one dimensional new collection frame work we can able
to store only values but not key, values.

Interfaces:

1. Collection
2. List
3. Set

Note: All predefined classes and interfaces of entire collection framework are available in a predefined package called util.

Differences between Collection, List and Set.

Collection	List	Set
* It is super interface of both List and Set	* It is a sub interface of collection interface	* It is also a sub interface of collection interface.
* Here we can store duplicate elements.	* Here also we can store duplicate elements	* Here we cannot store duplicate elements
* Here there will be no order of the elements.	* Here it will follows insertion order.	* It also won't follows any order
* Here we can able to retrive the elements in only forward direction (Top to bottom).	* Here we can able to retrive the elements in both forward and backward direction	* Here we can able to retrive the elements in only forward direction.
* Here we can able to add the elements only at ending of the index.	* Here we can add the elements any where based on requirement	* Here also we can add the elements only at ending of the index.

Collection: It is a predefined interface available in Util package.

Method

Methods:

→ Public int size()
→ Public boolean isEmpty()
→ Public boolean add(object)
→ Public Void remove(object)
→ Public void clear()
→ Public Iterator iterator()
→ Public object get(int)
→ public boolean addAll(object)

Iterator: It is a predefined interface available in util package, it will works like a Cursor.
* By using this we can able to retrive the values in only forward direction and by using this we can able to remove the elements also.

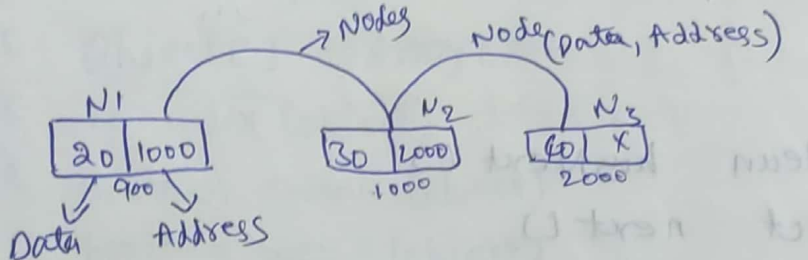
Methods:

→ Public boolean hasNext()
→ Public Object next()
→ public void remove(object)

List: It is a predefined interface available in a predefined package called util.
* It will follows insertion order and it will allo duplicate values
* For this interface we have two implementation classes like
1. Linked List
2. Array List

1. Linked List: It is a predefined class available in a predefined package called util.

* It will follows insertion order and it will allows duplicate values.
* Linked list will stores the elements in the form of nodes.
* Every node contains both data and address.
* Whatever the node containg the address, it is the actual address of the next node.
* If we find any node containg only data but not address then it will signifies that is the last element of the linked list object
* Whatever the values we will store, all those values will be stored in the form of object



Constructors:

Linked list ( )
Linkedlist (object)

Methods:

-> Public int size ( )
-> Public boolean isEmpty ( )
-> Public boolean add (object)
-> Public Object getLast ( )
-> Public Object getFirst ( )
-> Public void add (int, Object)
-> Public void addAll (object)
-> Public Iterator iterator ( )
-> Public ListIterator listIterator ( )

→ Public Object[] toArray()
→ Public Object get(int)
→ Public void addFirst(object)
→ Public void removeFirst()
→ Public void addLast(object)
→ Public void removeLast()
→ Public void remove(object)
→ Public void removeAll()
→ Public void clear()
→ Public boolean contains(object).

List Iterator:

*It is a predefined interface available in util pac
the main purpose of this interface allows us t
get the values in both forward and backward
directions.
* This interface will works like cursor.

Methods:

→ Public boolean hasNext()
→ Public object next()
→ Public boolean hasPrevious()
→ Public Object previous()

ArrayList:-

*It is a predefined class available in util package
* ArrayList will follows Dynamic Array to store or
retrieving the elements.
* Arraylist will takes less amount of time to get
values.
* ArrayList will follows insertion order.
*It will allows duplicate values.

Constructors:

ArrayList ()
ArrayList (object)

Methods:

- public int size
- public boolean isEmpty ()
- public boolean add (object)
- public void addAll (object)
- public void add (int, object)
- public void set (int, object)
- public void remove (object)
- public void clear ()
- public Iterator iterator()
- Public ListIterator listIterator()
- public Object get(int)
- public Object[] toArray()
- public int index (object)
- public boolean contains(object)
- public boolean equals (object)

Set:-

* It is a predefined interface available in util package.
* It does not follows any order.
* Set won't allows duplicate elements.
* For this interface we have three implementation classes like HashSet, LinkedHashSet, TreeSet.

HashSet:

It does not follows any order and it won't allow duplicate elements, Here we can store null values

Constructors:

HashSet()

HashSet(object)

Methods:-

→ Public int size()
→ Public boolean isEmpty()
→ Public boolean add(object)
→ Public void addAll(object)
→ Public void remove(object)
→ Public Iterator iterator()
→ Public Object[] toArray()
→ public void clear()

LinkedHashSet

*It is also a predefined class present in Util package.
*It will follows insertion order and it will allows null values and restrict duplicate elements.
*It will follows LinkedList and set also.

Constructors:

LinkedHashSet()
LinkedHashSet(object)

Methods:-

→Public int size()
→Public boolean isEmpty()
→Public boolean add(object)
→public void addAll(object)
→public void remove(object)
→Public Iterator iterator()
→Public Object[] toArray()
→Public void clear()
→public Object get(int)
→public void add(int, object)
→Public ListIterator listIterator()

TreeSet:- It is the implementation class of SortedSet, NavigableSet and Set interfaces.

* It won't allows duplicate values why because it will compare each and every values by using comparator and then it will display all the values by default sorting order (by default ascending order).

Constructors:

TreeSet()
TreeSet(object)

Methods:-

- Public int size()
- public boolean isEmpty()
- public boolean add(object)
- public void addAll(object)
- public Object first()
- public Object last()
- public void remove(object)
- public Iterator iterator()
- public boolean contains(object)
- public void clear()
- public Object get(int)
- public Set descendingSet()
- public boolean equals(object)

MAP:- It is a predefined Interface present in util package, by using this we can able to store (key, value) pairs.

* Here keys cannot be duplicate and values might be duplicate.
* Here we can able to store null keys and null values.

* For this interface we have three implementation classes like HashMap, LinkedHashMap and TreeMap.
* It does not follow any order of the elements.

HashMap:

It is a predefined class present in util package, will allows us to store the elements in terms of (key, value) pair.

* Here also keys cannot be duplicate and values might be duplicate.
* Here it does not follows any order of the elements.
* Here we can store single null key and multiple null values.

Constructors:

HashMap()
HashMap(Object, object)

Methods:

→ Public int size()
→ Public boolean isEmpty()
→ Public boolean Put(obj, obj)
key value.
→ Public void putAll(object)
→ Public void remove(obj)
key
→ Public void clear()
→ Public Set entrySet()
→ Public Set keySet()
→ Public boolean equals(obj).

LinkedHashMap: It will follows insertion order. it is the implementation class of LinkedHashSet, Map.

Tree Map: It is also a predefined class present in util package.

* It will stores the values in terms of (key, value) pairs, Here we cannot store even single null keys.
* It will follows sorting order by default ascending order.

Constructors:

TreeMap()
TreeMap(obj, obj)
TreeMap(size)

Methods

Public int size()
Public boolean isEmpty()
Public void put(obj, obj)
Public object get(obj)
public Iterator iterator()
Public Set entrySet()
public Set keySet()
Public NavigableSet descendingSet()
Public void clear()
Public void remove(object)
public boolean equals(object)

Legacy collection Frame work:

*It will contains Data Structures concept and it is classified into two types.

1. 1 Dimension Legacy collection Frame work
2. 2 Dimension Legacy collection frame work.

1 Dimension Legacy collection Frame work:

* Here we can able to store values any it contains both classes and interfaces.

Interfaces:
Enumeration
Queue

Classes
vector
Stack
Priority Queue.

Enumeration: It is a predefined interface present in util package, It will works like a cursor in legacy collection frame work, by using this we can able to retrive the elements of both classes and interfaces of Legacy Collection frame like Iterator in new collection frame work.

* It is inbuilt Synchronised (thread Safety).

Methods:
Public boolean hasMoreElements()
Public Object nextElement()

Queue:-
It is a Predefined interface available in util package.

* It will follows FIFO order.
* For this interface we have to provide functionality either by using "Priority Queue" or "Linked List" classes.

Profile:-
Methods:-

* Public int size()
* Public boolean isEmpty()
* public void add(Object)
* public void addAll(Object)
* public void clear()

→ public void remove (obj)
→ public Enumeration elements()
→ public void add (int, obj)
→ public Iterator iterator()

Vector:

It is a predefined class, it will follows insertion order as it follows array list class.

* The default capacity of vector is 10 (elements)

profile:

constructor:

vector()
vector(object)
vector(int)
↓
capacity.

methods:-

→ public int size()
→ public int capacity()
→ public void add ~~(object)~~ (int, object)
→ public void addElement ~~(int, object)~~ (object)
→ public void remove (object)
→ public boolean equals (object)
→ public boolean contains (object)
→ public Enumeration elements()

Stack:

* it is a predefined class, it will follows LIFO order.
* we can perform push and pop operations, by using push method we can store the value, by using pop method we can remove the value.
* we can able to search any element in the stack and we can return the index number of that element

Profile:

Constructors:

* Stack ()
* Stack (object)
* Methods:

→ public boolean isEmpty()
→ public int size()
→ public void add (object)
→ public void remove (object)
→ public void push (object)
→ public object pop()
→ public int search (object)
→ public object peek()
→ public Enumeration elements()

Dictionary:

* It is a predefined abstract class present in util package
* It will stores the value in terms of (key, value) pair
* It does not follows any order of the elements
* Here we can store null keys and null values.

Hashtable:

* It is a predefined concrete class present in util package.
* Here we can store the values in terms of (key, value) pair
* It does not follows any order of the elements.
* Here we can store single null key.

Profile:

Constructors:

Hashtable ()
Hashtable (obj, obj)

Methods:

public int size()
public boolean isEmpty()
public void put(obj, obj) (key, value)
public void putAll(obj)
public void remove(obj, key)
public void replace(obj, value) (key)
public void replace(obj, ov, nv) (key, oldvalue, newvalue)
public Set entrySet()
public Set keySet()
public void clear()
public boolean containsKey(obj)
public object get(obj) (key)

Properties:

* It is a predefined class present in util package.
* By using this we can able to store the values in external file like either properties file (or) .rbf file (rbf → resource bundle file). in terms of (keys, value) pairs
* Here we need not to store the values in Sourcecode, if we store the values in source code, if we want to perform any changes with respect to data then we need to perform the changes in Sourcecode
* If we perform Multiple changes in Same Sourcecode then there will be impact on source code and their results (or) some times the entire application will not respond.
* But by using properties file we can avoid these drawbacks.
* Here we will store the values in external file then we will get the data from external file to Sourcecode

and display the values.

* If we want to perform any changes then we can avail disturb (or) change external file but not source

* Profile:

Constructors:

Properties ()

Properties (String key, String value)

Methods:

Public void setProperty (String key, String value)

Public String getProperty (String key)

Public void load (object)

Public Enumeration values ()

Steps (or) guidelines to deal with properties class:

* Initially we need to create external file, store the values in terms of key = value pair then save that file as either .Prop (or) .properties (or) .xbf.

* Now we need to create Java application by importing necessary predefined packages like (io & util)..

* Now we need to open external file in read mode either by using FileInputStream class (or) FileReader class by providing external file name as a parameter with respect to complete path.

* Now we need to create object for properties class.

* We need to load external file with respect to properties class object.

* Now we need to get the values through properties class object by providing respective key names which are