

Language:

Language is a communication channel between two end users.

It is used to transfer the information from between two end users.

Many of the people use the languages like Hindi, Telugu, Tamil, English for communication.

If we want to interact with the machine, then we need one special language that is Programming Language.

---> By using above languages we cannot interact with machine so we need special language like PROGRAMMING LANGUAGE.

PROGRAMMING LANGUAGE:

A language which is used to make communication between end user to machine is called PL.

PL is a installable softwares.
These are provides basic information(raw material).
By using these basic information(PL) we can develop our own applications,technologies,tools,frameworks,Operatingsystem.
ex: c,c++,c#,java

Programming Language:

A language, which is understood by the machine to process is called Programming language.

Aspects of Programming Languages:

- * Storing the data
- * Accessing the data
- * Processing the data
- * Delete the data
- * Update the data.

We have three types of programming languages.

- a. Binary language or Machine language.

- b. Assembly language
- c. High-level language.

Programming Language provides basic information or raw material to develop our own applications.

Programming Languages provides syntax (rules) or semantics (structure).

Binary -Language:

This is first and machine level language.

This is only understands by machine.

Machine can understand the data only in the form of zero and one's.

Each and every character first converted into ASCII / Unicode values later it is converted into zero and one.

Developing the program under binary language is very difficult and time consuming.

The main drawback of binary language is platform dependency.

Developing the application on binary language will take more time.

If application development time is increase automatically application development cost is also increases.

Platform:

It is the combination of software and hardware, it provides environment, to execute the programs.

Executing the programs means install software, update software's, typing something in notepad.

Platform Dependency:

If we are developing and compile the program under one operating system, the same program is not executing under different operating system is called platform dependency.

Assembly language:

This is low level language. It is used to avoid drawbacks of binary language. This language can avoids hard coding (complex), but unable to avoid platform dependency.

This language internally uses MNEMONICS (ADD, SUB, MUL, DIV) for developing the program.

High-Level Language:

These language are very user friendly language to easy to develop and easy to understand.

We have different languages under high level languages. Like C, C++, .Net, Java, COBOL, Pascal and etc.....

Java is the language to avoid the problems of platform dependency.

With the C, C++ we cannot develop internet support application and platform independency applications.

With the help of java we can develop internet support applications. The reason java is pure platform independent software.

Software:

It is a development tool, which is used to convert our imaginary things to real world existing things, by writing set of programs.

We have three types of software's.

System Software:

The software, which is used to develop hardware functionalities.

Ex: C, VC++, Embedded Systems, USB Driver Software's, Compilers, Translators, Printer Software's, Scanner Software's.

Application software:

All the back end or database software is comes under application software.

Ex: Oracle, MySql, Sql, DB2, SqlServer.....

Internet software:

To develop internet applications or internet support applications.

Ex: Java, .Net.

Java is coming in the form of three flavors.

- Java Standard Edition.
- Java Enterprise Edition.
- Java Mobile or Micro Edition.

Java software comes with two separate components.

- a. JDK
- b. JRE

Java Buzzwords or Java Features:

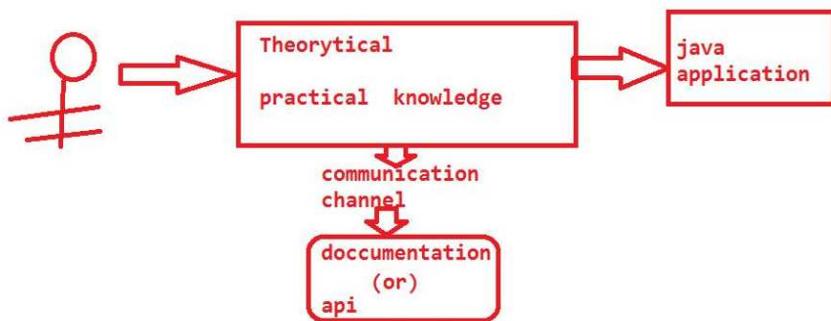
- Simple
- Platform Independence
- High Performance
- Security
- Architectural Neutral
- Portable
- Robust
- Multithreading
- Distributed
- Dynamic
- Architectural Neutral
- Portable
- Oops

Simple:

Java is a simple language, due to the following reasons.

1. Garbage collection.
2. Provides huge predefine code. (Packages- rt.jar).
3. Documentation.
4. User Friendly Syntax.

Documentation:



Garbage Collection:

Java provides great services like Garbage Collector to clean unused memory or unreferenced memory.

This is also called as finalization.

If we go for other languages, programmer should be writes the code for both memory allocation and de-allocation.

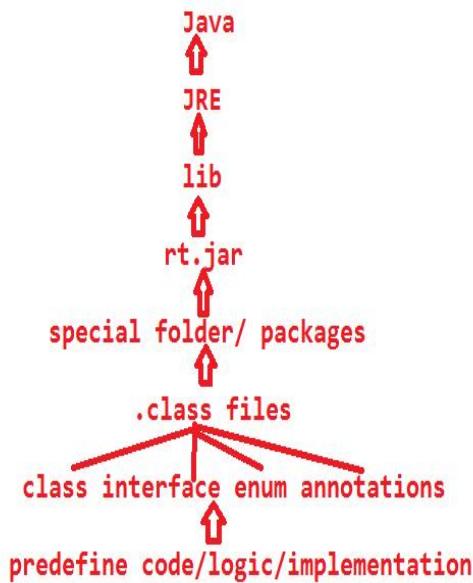
But in java, programmer is not writing any code for de-allocating the memory, JVM itself communicating with garbage collector to clean-up the memory by using garbage collector.

Java provides user friendly syntax means programmer can easily understand and remember more time, and easily placed those syntax's in developing of projects.

Java provides a huge predefine code or low level logic.

With the help of this code, we can develop our project within the less time and very easily.

All Java Standard Edition predefine logic is coming to our machine mean while of installation in the from archived file that is rt.jar

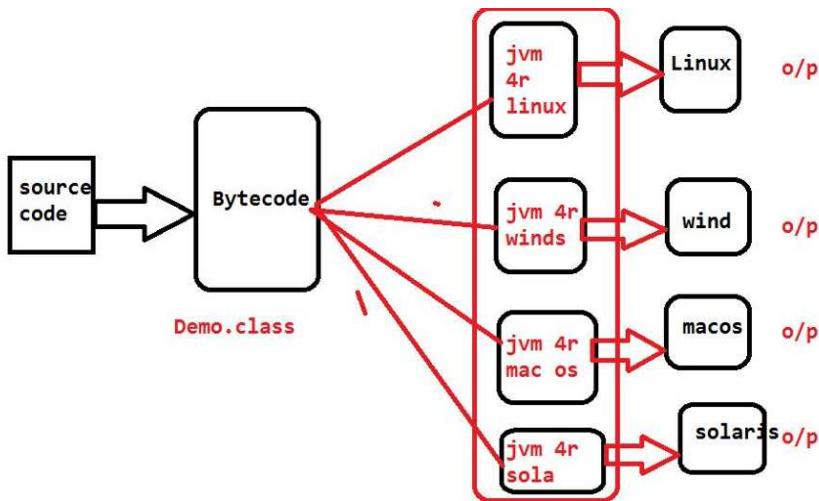


By Reading of java provide documentation, we can understand each and every java functionalities.

The syntaxes which are given by the java software, those are very user friendly.

Platform Independence:

Developing an application, compile the same application in one operating system, if we execute the same application in all operating systems is called platform independency.



Source-Code:

The code which is writing in any language (C, C++, .Net, Java) that code is called source code.

Machine code or Binary code: The code which is understands by the machine is called binary code or machine code.

Source code is not understands by the machine. So we need translate source code to machine code. To translate the source to machine code, java provides two translators.

1. Compiler
2. JVM

Translator: It is program which is used to convert from one type of code to another type of code.

Every java name must be ended with .java file, once we save the file, then that file available in hard disk.

Within the .java file we can write multiple classes.

If we want to interact with the compiler we need one predefine development tool that is "javac".

"javac" always followed by FileName.java

FileName.java must be contains source code.

Ex: javac FirstDemo.java

Whenever we written the above syntax in command prompt automatically, compiler will come in to the picture, it will interact with .java file and finally compiler will communicate with hard disk.

If the file not available compiler will give one error like file not found error.

If available ".java" file is loading from secondary memory to primary memory.

Mean while loading compiler will check is source code properly designed under java grammar rule or not, if not compiler will give Compile Time Error.

If code is properly designed then compiler will convert into byte code or JVM understandable code or intermediate code, and finally that code will be placed into ".class" files.

As many classes we have in one ".java" file, those many ".class" files will be created by the compiler.

Different class components are having different ".class" files and different byte code.

After creating the ".class" files, compiler will place all the ".class" into same directly (folder) in hard disk.

Bytecode:

The code which is generated by the compiler is called bytecode. It is not understand by the programmer and machine.

Bytecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code. This byte code is a machine independent code. It is not a completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by JVM. Bytecode is a machine code for JVM. But the machine code is platform specific whereas bytecode is platform independent that is the main difference between them. It is stored in ".class" file which is created after compiling the source code. Most developers although have never seen byte code (nor have ever wanted to see it!) One way to view the byte code is to compile your class and then open the ".class" " file in a hex editor and translate the bytecodes by referring to the virtual

machine specification. A much easier way is to utilize the command-line utility javap. The Java SE DK from Sun includes the javap disassembler that will convert the byte codes into human-readable mnemonics.

Compilation: Converting from source code to bytecode.

Bytecode:
flexibility -->understand by any type of
jvm(windows,solaris,mac os,linux)

platform independent code.

this bytecode will convert into
any type operating system understandable code.

for this purpose we required different jvm.

The code which is available in “.class” file, that code representation is always in the form of bytes.

Whatever the code is generated by compiler, that is not understood by the machine. So again we need to convert from byte code to machine code, for that purpose we need one more translator that is JVM.

If we want to interact with JVM we need one development tool that is “java”.

“java” command always followed by class name.

Ex: java Demo

Whenever we write above syntax in command prompt then JVM will come into the picture, and interact with “.class” file name, and finally loaded from secondary memory to primary memory. If byte code is properly organized then we will get executable code that code is executing under all operating system. Then we can say java is a platform independency language.

But JVM is pure platform dependent component.

The reason is we have different JVM'S.

Whatever code is generated by compiler that code is understood by all JVM'S. Those JVM'S again convert and executed in their dependent operating systems.

Execution: Converting from byte code to executable code.

Slogan:

Write Once and Reuse Anywhere (WORA)

Write Once and Run Anywhere

Architectural Neutral:

Developing and compile the application under one processor, executing the same application under different processors is called Architectural neutral.

Portable:

The combination of Platform Independence and Architectural Neutral is called portable.

OOPS:

OOPS provides some rules and regulation and designs which help to develop application very easily.

We have the following OOPS principles. Those are

- 1) Class
- 2) Object
- 3) Encapsulation
- 4) Abstraction
- 5) Inheritance

6) Polymorphism

Q) Is java software independent or not?

**A) Java software is platform dependent, the reason is we have different java softwares for different os.
One os support java software is not executing in other os.**

Linux ARM 32

Linux x86

Linux x64

Mac OS X

Solaris SPARC 64-bit

Windows x64/

Q) Is java source code platform independent or not?

A) What ever the program we writing under any operating system java syntax are never be changes so source code is PI.

Q) Is java bytecode platform independent or not?

A) java bytecode can be run on all os. so java bytecode is PI.

Q) Is java compiler platform dependent or not?

A) NO. Java compiler is platform independent, the reason is what ever the code we writing in the java that will common for all os, so all os java compiler architecture is same. so compiler is platform independent.

Q) Is java project/application PI or not?

A) Java project or applications are PI only, the reason is java application can be running under all os.

Q) Is jvm platform independent?

A) No. One jvm is not suitable for converting from bytecode to os understandable code we required different jvm. so jvm is not a platform independent. It is pure platform dependent component.

History of Java:

Development of java is start from Green Team (Java Team).

Initially java had introduced for digital devices (set-top boxes, televisions etc).

Now java is used internet programming, mobile devices, games, e-business).

James Gosling, Patrick Naughton introduced in 1991 June.

Initially it was called Greentalk and file extension is ".gt".

Later this language renamed as OAK.

OAK is symbol of strength.

OAK is the tree name.

It is the national tree for Germany, USA, France

In 1995 OAK renamed as Java.

No abbreviations for OAK and Java.

Java is an island of Indonesia.

This island is famous for coffee bean.

In 1995 Java introduced version like JDK Alpha and Beta.

In 1996, JDK 1.0 version released in the market.

JDK Alpha and Beta (1995)

JDK 1.0 (23rd Jan, 1996)

JDK 1.1 (19th Feb, 1997)

J2SE 1.2 (8th Dec, 1998)

J2SE 1.3 (8th May, 2000)

J2SE 1.4 (6th Feb, 2002)

J2SE 5.0 (30th Sep, 2004)

Java SE 6 (11th Dec, 2006)

Java SE 7 (28th July, 2011)

Java SE 8 (18th March, 2014)

Java Software released in the market in the form of three flavors/editions.

Java Standard Edition

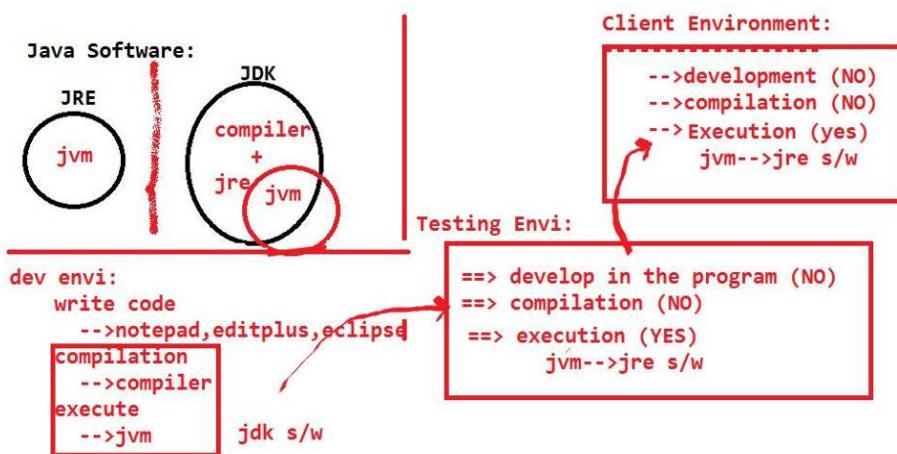
Java Enterprise Edition

Java Mobile/Micro Edition

Java software come into the market in the form of two flavours.

1. JDK

2. JRE



Why java is open source?

- > Download/install java soft ware.
free of cost for downloading from networks/internet
- > Developing project on java.
Notepad, Editplus, Eclipse,
(Text editors)
- > Developing project within the less and performance
(Frameworks--> Spring, Hibernate, Structs)
- > Running project on Server.
Apache Tomact, GlasssFish,.....
- > To know internals of java software.
(source code)

JDK:

==

java standard edition development kit. It contains compiler and jre.it will provide:

- > Developments tools
- ==> source code
- > public jre
- > java db

Q) Why java is open source?

- * Java related software like JDK and JRE are freely available in the network.
- * Java software development code is also freely avialable in the market.
- * Java supported softwares (Servers(Apache tomcat), IDE's (Eclipse), Frameworks(Spring,Hibernate,Structs)) are also freely avialble in the market

With the help above flavors we can develop the following applications.

1. Standalone/Desktop applications.

2. Client-Server programming (socket programming)
3. Database interaction applications.
4. Web Applications.
5. Enterprise/Distributed applications.
6. Interoperable applications.

High Performance:

Application performance depends upon the following things.

- 1) Less response time
- 2) Less memory utilization
- 3) More end users.

If we go for other languages, they are using only one translator for executing the program so those languages will take more time to execute. If any application will take more time to execute that application performance will be decreased.

To avoid this problem java internally uses two translators

1. Interpreter.
2. JIT Compiler

Compiler:

It is a translator. It will convert our source code to byte code. It is check all line at a time, if any syntax errors available first those errors will be placed into memory, after checking all the statements finally it will print all error information on the output device (console).

Interpreter will translate from byte code to machine understandable code. It will check line by line. If current line valid then control goes to next line otherwise it will print error or exception message on the console.

Interpreter is very good at execute the single time execute statements.

Whereas JIT compiler good at execute the looping statements.

But these two components don't have any capability to understand whether the statements are single time execute or looping statements.

In JVM, there is one special component i.e. "hotspot profiler".

This hotspot profiler will recognize the statement behavior.

If statement is single time execution statement then those statements will give to Interpreter, if statements are looping statements then those statements are handover to JIT compiler.

In java both Interpreter and JIT compiler works together and execute the program within the less time, if program is execute within the less time automatically the performance of an application will be increase.

By this reason, we can say java is a high performance language.

Robust:

Java is a robust language due to the following reasons.

- 1) Proper Memory Management.
- 2) Exception Handling.
- 3) Casting.

1) Proper Memory Management:

Java internally used one background program, which is used to maintain memory properly.

The background program is Garbage Collector. This Garbage Collector, will check is there any unused memory or not, if yes that memory will be cleanup, the same memory will be allocated to other data.

2) Exception Handling:

Java is giving very great support to handle the both compile time and runtime exceptions.

When ever exception is raised, internally java uses some predefine code to generate exception message in the following manner.

In the exception message it will give information about

- 1) File Name Information.
- 2) Class Name Information.
- 3) Method Name Information.
- 4) Line Number Information.
- 5) Pre/User Define Exception Class Information.
- 6) Description of an exception (Exception Message)
- 7) Package Information.

By using above information programmer can resolve error and exception within the less time with our facing complexity.

3) Casting:

In java both compilation time and runtime both compiler and JVM will check variable type and range and value.

By above three advantages, we can say java is a robust language.

Secure:

Java provides huge security in the following ways.

- 1) Avoiding the pointers.
- 2) Packages
- 3) Security Manager
- 4) Verifier.

If we are avoiding the pointers we can provide security, avoid virus and hacking code.

- ➔ With the help of packages we can provide security to both default and protected data.

Packages will provide full security to default data, but packages are not provides full security to protected data.

If we are doing some modification on protected data, we can access from outside of the package also.

Those modifications are made as static and other package class is the sub class of current package class.

- ➔ Verifier will check whether the byte code is properly organized or not and is there any virus and hacking code or not.

If yes verifier provides one error like `java.lang.VerifyError`.

Distributed:

Whatever applications developed under the java, those application services are equally distributable/sharable to all end users.

Data is shared between all the machines in the form of object within the same time.

Dynamic:

With the help of java we can develop dynamic pages and animated games and allocates the memory dynamically.

Static Applications:

The application which always provides same response to all end user are called static applications.

Dynamic Application:

The applications which are always provides different response to different endusers at different execution time are dynamic applicaitons.

Note: With the help of java we can develop both dynamic and static applications.

Memory allocation:

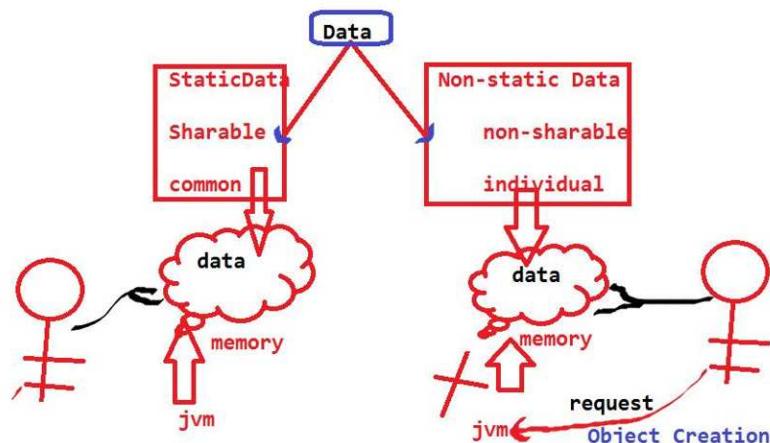
Java supports two types of memory allocations.

Static memory allocation:

By default jvm will provides the memory for all static data.

Dynamic memory allocation:

When we required memory for non-static data, then only we can make request jvm through object creation for allocating memory for non-static data.



Multithreading:

Executing the multiple threads concurrently is called multithreading.

With the help of multithreading we can execute our programs with the in the less time, then automatically we will improve the performance of an application.

This concept will provide proper co-ordination between gaming elements.

In general our java program contains two types of threads

- 1) Non-Daemon (MainThread).
- 2) Daemon (Garbage collector).

With the help main thread we can call methods, we can create memory.

With the help of garbage collector we can clean-up the memory.

Object Oriented Programming System:

Any language which has been developed based on object oriented programming system principles those language are called OOPL.

Ex: C++, .Net, java.

We have the following OOPS.

- 1) Class.
- 2) Object.
- 3) Encapsulation.
- 4) Abstraction.
- 5) Inheritance.
- 6) Polymorphism.

Major and Minor versions:

Java software comes in the market in the form of two versions.

- 1) Major Version.
- 2) Minor version.

Major Version: If we want to add new features to the existed software, then we should go for major version.

Ex: jdk 1.6.0

jdk 1.7.0

Minor version:

If we want add bug fixing code to the existed software then we can release our software as an minor version.

Ex: jdk 1.6.0_20

One java software is not suitable for all the operating systems. Different operating systems have their own java software.

Environment Variables Setting:

Command prompt is comes with operating system, whereas java development tools (binary files) and library files are comes with java software.

So our command prompt is unable to recognize the development tools.

Then programmer should provide the information about binary and library files to command prompt, with the help of environment variables.

OS by default uses environment variables to recognize software.

For java we have two types of path settings.

1) Temporary settings.

2) Permanent settings.

Temporary Settings:

Whatever the setting which we are done at one command prompt those settings are applicable to that command prompt only not applicable to other

command prompt and also if we did close the command prompt automatically all the settings are gone.

Syntax: set path=C:\Program Files\Java\jdk\bin;

With the help of above settings our command prompt recognize the binary files. (javac, java, javap command).

Syntax": set classpath=.;C:\Program Files\java\jre\lib\rt.jar;

With the help of above settings our compiler and jvm can recognize the predefined .class files.

Binary files used to compile and execute the program, where as Library files are used develop the user define program.

Permanent Settings:

The settings which are applicable for entire system, those settings are called permanent settings.

Steps to permanent settings:

Right click on my computer Properties Advanced System Settings

Advance Environment Variables under System variables.

Click on new button set the path.

Variable Name: path

Variable Value: C:\Program Files\Java\jdk1.6.0\bin;

Click on new button set the classpath.

Variable Name: classpath

Variable Value: .;C:\Program Files\Java\jre6\lib\rt.jar;

Files:

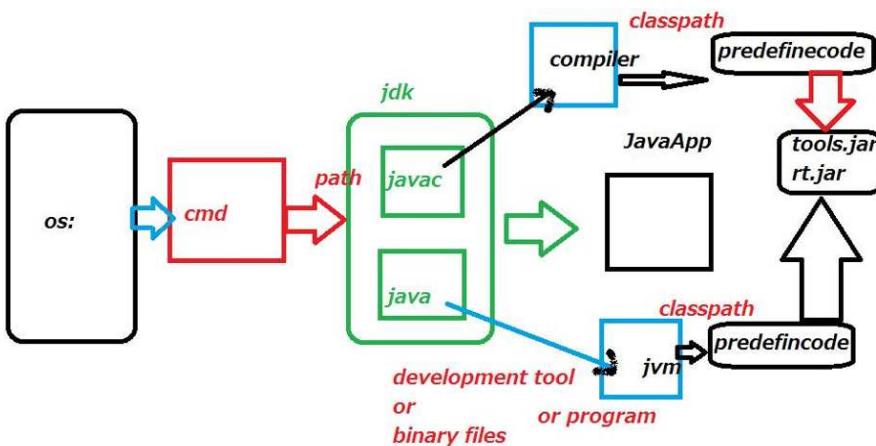
path

Binary Files: Executing the code in sequence order.
 check code whether valid or not
 compiler the code(sc--bc)
 execute the code/interupted
 decompile(bc-sc)

C:\|PF\|J\Jdk\bin;

Library Files: **classpath**
 (predefine code) we can also called as supporting files
 -->programmer develop new code.

C:\|PF\|J\Jre\lib\rt.jar;



Object:

It is a real world existing thing.

It provides memory for non-sharable data or non-static data or instance data.

Object is having following characteristics.

Identity: To differentiate the object to another

(Differentiate one memory to another memory).

Properties: To hold the information (variables, Fields, Instances, Member Variables, State).

Functionalities: to hold the logic (to do some operations on properties). (Methods, Operations, Behavior, Functions).

One object is communicating with another object with the help of functionalities.

Object having *it is real world existing thing, it provides the memory for*
3 characteristics: *non-sharable/non-static/individual data.*

- 1) **identity:** differentiate one object to another object
to differentiate the one memory location to another memory location.



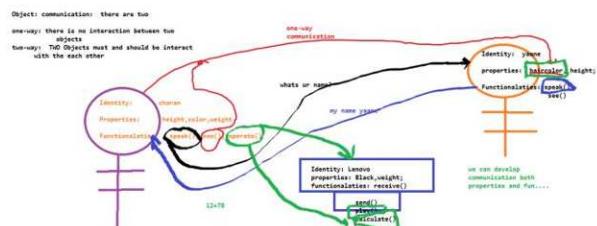
- 2) **properties** → *to hold information-->accno,accholdename,pin,amount*

Variable/field/state/instances

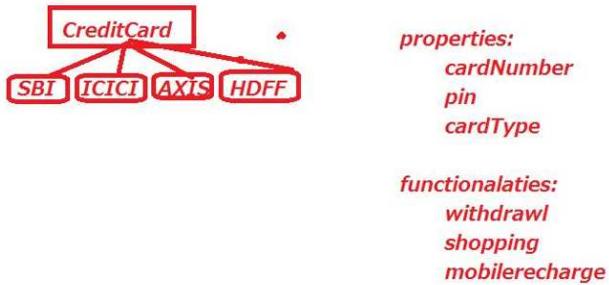
- 3) **functionalities** → *to hold logic*

Methods/operations/behaviour

to do some operations on properties



real time example for class and object:



Class:

It is a imaginary thing, it is contains to hold object information (identity + functionalities + properties).

Class is a combination of

Variables + Methods or

State + Behavior or

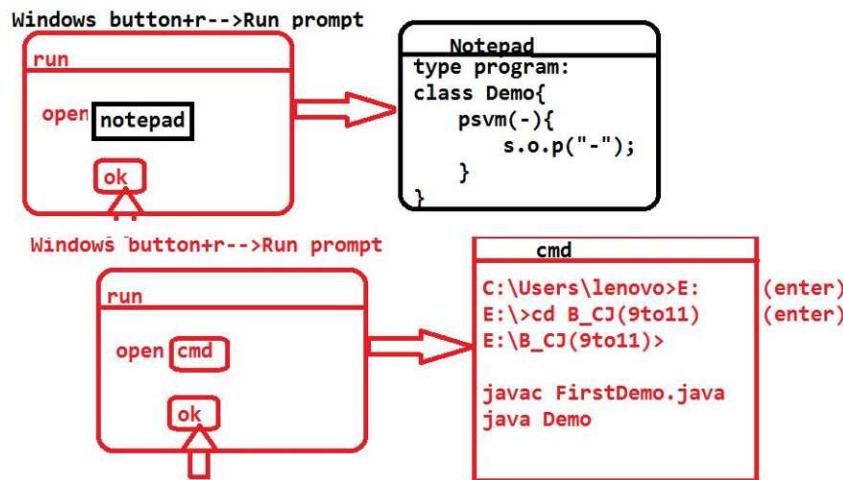
Member Variables + Member Methods

It is a model.

It is a design.

It is a blueprint.

How to open notepad and command prompt and compile and execution:



First java program:

```
class Demo{

    public static void main(java.lang.String[] ram){

        java.lang.System.out.println("hi friends");

    }

}
```

Q) Why main is public?

JVM is one program. It is located in one package. Our program/class available in one more package. So one package data want to communicate with another package data we should use public keyword.

Q) Why main method is static?

Without providing memory by programmer with the help of JVM, for main method, JVM itself want to provide the memory for main and calling purpose, the method (main) must be static.

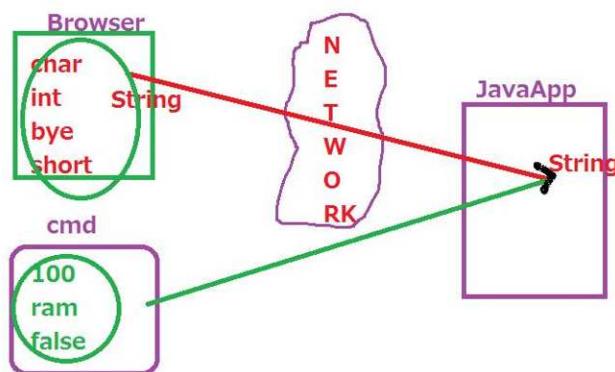
That means without creation object if we want to call the main method by using class name by the JVM, we mention the main is static.

Q) Why main method is having void?

Main method is not giving any value to JVM.

Q) Why main method is having String [] parameter?

The data which is coming from browser or command prompt that data is always string format. To holding that data we need String [] variable.



File Name: FirstDemo.java

Class Name: Demo

Compilation: javac FirstDemo.java

Execution: java Demo

If we are doing any modification to source file programmer must be save file and compile the program again.

Valid syntax:

String [] ram;

String [] ram;

String []ram;

String ram[];

Invalid syntax:

```
[] String ram;
```

Note: we cannot place array symbol before data type.

```
class Demo{  
    static public void main(java.lang.String[] ram){  
        java.lang.System.out.println("hi friends");  
    }  
}
```

Note: We change places between static and public.

```
class Demo{  
}
```

We can develop empty .java file. This file can be compiled but we cannot get any ".class" file, the reason is there is no source.

We cannot execute the program. The reason is there is no class name.

We can create empty java class. We can compile but we cannot execute.

In java 1.6 we got one error: NoSuchMechError:main

```
class {  
}
```

We cannot develop class without class_name.

If we write the following code we will get one compile time error that is identifier expected.

```
class Demo{
```

```
public static void main(String [] args){  
    System.out.println ("java program")  
}  
}
```

Every statement must be ended with semicolon, otherwise we get compile time error that is ';' expected.

```
class Demo{  
    public static main(String [] args){  
        System.out.println ("java program");  
    }  
}
```

In java method must be have return type. If we are not writing, we will get compile time error that is invalid method declaration, return type required.

```
class Demo{  
    static{  
        System.out.println("this is static block");  
        System.exit(0);  
    }  
}
```

Without main method also we can compile and execute the program from java 1.0 to java 1.6. But in java 1.7 and 1.8 main method is mandatory.

System.exit(0) means, we are explicitly stopping the program.

```
class Demo{  
    static public void main(String... ram){
```

```
        System.out.println("this is main method");

    }

}
```

(...) called var args (Variable arguments). We can also calls elipse.

Internally var args maintain array concept only.

This feature is introduced in java 1.5.

Valid Syntax:

```
String...ram;  
String... ram;  
String ...ram;  
String ... ram;
```

Invalid Syntax:

```
String ram...;  
...String ram;  
String. ..ram;  
String.. .ram;  
String . . . ram;  
  
class Demo{  
    static public void main(String... ram){  
        System.out.println("this is main method");  
    }  
}  
  
class A{
```

```
}

class B{

}

class C{
```

Within the one ".java" file we can create multiple classes. As many class keywords we have those many ".class" file will be generated by the compiler.

In the execution time, we should give class name, which have main method.

```
class Demo{

    static public void main(String... ram){

        System.out.println("this is Demo main method");

    }

}

class A{

    static public void main(String... ram){

        System.out.println("this is A main method");

    }

}

class B{

    static public void main(String... ram){

        System.out.println("this is B main method");

    }

}
```

Within the one “.java” file we can write multiple classes and also in all the classes we can write main method also.

Which class name we are going to be give at execution time that class main method will be execute.

```
javac FirstDemo.java
```

```
java Demo
```

```
java A
```

```
java B
```

We can compile more than one .java file at a time with the help of following syntax.

```
javac *.java
```

But we cannot execute more than one “.class” files at a time directly.

If given .java file is not existed, then compiler will give file not found error.

if given .class file is not existed, then JVM will provide
NoClassDefFoundError: class name

File Name no need of same as class Name. But if the class is public type then our filename must be same as class Name otherwise compiler will give one compile time error i.e.

```
class <class-name> is public, we should declared file name as <class-name>.java
```

```
public class Demo{
```

```
}
```

In the above scenario we should save filename as Demo.java

We cannot write more than one public class within the one “.java” file as bellow.

```
public class Demo{
```

```
}

public class C{

}

*****
```

No user define/predefine method will be executed automatically in java, except main method.

If we want to execute the user define/predefine method then we should call explicitly.

```
class B{

    static void m1(){

        System.out.println("m1 method");

    }

    static public void main(String... ram){

        System.out.println("this is B main method");

        //m1();

    }

}
```

We can call main method explicitly by using below syntax.

```
main(new String[0]);
```

If we call main method of other class then we should use bellow syntax

```
Class_name.main(new String[0]);
```

```
class A{

    psv main(String... ram){
```

```
        System.out.println("hi");

        main(new String[0]);

    }

}
```

We can't call main method within the main method. This syntax will reach infinity loop.

Finally JVM will provide one runtime error is "java.lang.StackOverflowError".

At a time we can load only one .class file by using development tool like java, javaw, javaws.

If we want load other ".class" file we should use the following methodologies.

1. By using development tools (java A)(javac,javap,javaw)
2. By using class_name and calling static variables or static method(A.main(-)).
3. By using object creation (new A())
4. By using Reflection API (Class.forName(-))
5. By Inheritance
6. ClassName.class
7. De-serialization Procedure.

Compiler will check class information first in method later in class later in ".java" file later in package later in current working directly later finally control goes to predefine class.

Without main method also we can execute the program upto 1.6 not from 1.7 onwards.

Upto 1.6 jvm without checking main method exists or not first control give to static blcok later main method

But from 1.7 onwards jvm first checks whether main method exists or not. If exists, jvm will control to static block later main method

If not exists we will get runtime error.

```
public class Demo{  
    static{  
        System.out.println("sb2");  
        System.exit(0);  
    }  
}
```

Java Programming Elements:

The following are the java programming language elements (The syntax which is used to develop the program)

Those are

1) Comments.

2) Packages

3) Java Tokens

a. Keywords.

b. Literals.

c. Operators.

d. Separators.

e. Identifiers

i. Class

1. Methods

2. Variables

ii.Interface

 1.Methods

 2.Variables

iii.Enum

 1.Methods

 2.Variables

4) Annotations

5) Constructors

6) Blocks

7) Import statements

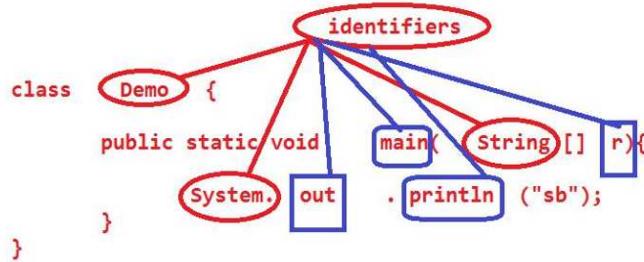
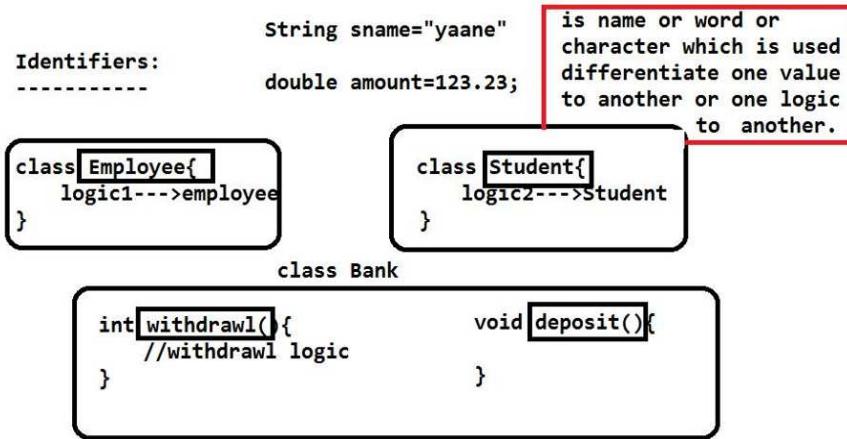
Identifiers:

It is name or word or character which is used differentiates from one java element to another java elements.

```
class Student{  
    public static void main(String... ram){  
        System.out.println("hi");  
    }  
}
```

To differentiate from one java element to another java element, then we can go for identifiers'.

Identifier is a name, character, word which is differentiate one java element to another java element.



Naming Conventions in Java:

1) Class:

Class name must be starts with capital letter

Ex: Student

If any class name having multiple words first word first letter must be capital letter, the remaining words first letter must be capital letter.

Ex: StudenPersonalDetails

We can use digits in the class name

Ex: Student1;

But don't start class with digits.

We can use two special characters with in the class name.

ex: Student_Info

Ex: Student\$Details

We can start class name with the help of '_' and '\$' and combination also.

Ex: __Student

\$_\$Bank

We can use predefine class name as user define class name. But in this time we cannot get functionalities from predefine class.

Ex: class String{

}

Note: Don't use predefine class names as user define class names.

Don't use keywords as a class names.

Invalid Class Names:

Class name never be start with digits.

#ex: 1Student //wrong

Class name does not have any spaces.

Ex: Student Details//wrong

Student Personal Details//wrong

Class name doesn't have any special characters except under(_) and dollar(\$).

Ex: Student"Info //wrong

Ex: Student=Info //wrong

We cannot use keyword for class names.

```
Eex: class while{//wrong  
}
```

Whatever the rules we have int java about to classes, the same rules applicable to interfaces, enum, annotations.

Keywords: Keywords must be write within the small case.

Ex: for, if, else, private, public

Ex: For, Native, Strictfp //wrong syntax

Variables:

Variables are always starts with small letter.

Ex: String name;

byte age;

float sal;

Variables name having more than one word then first word first letter must be small letter, the remaining words first letter must be capital letter. The remaining rules same as class rules.

Ex: String studentName;

String accountHolderName;

Methods:

Whatever the rules we have related to variables, the same rules we have for methods also.

There is a small different between variables and methods is method always ended with '(' and ')'.

Constants:

All the final variables are comes under constants in java

Ex: final double PI= 3.14;

final variables or constant variable names are always in the capital letters.

Ex: MIN_PRIORITY1

MAX_PRIORITY10

NORM_PRIORITY5

Packages:

Package names are must be in the small case.

Ex: java.lang

java.util

java.io

java.net

com.ram

com.nit

com.google

Literals: A values which is assigns to variable is called literal

Ex: int a = 10; //10 is an int literal

char c = 'b'; //b is an char literal

boolean b = false; //false and true are boolean literals

boolean b1 = true; //true--boolean literal

String s = "ram"; //ram-- String literal

Student s = null; // null---is also called as literal

double d = 23.34d; //23.34---is an double literal
char literal is always within the single quote.

Literals: The value which is assign to variable.

1)null
2)false
3)true

predefine literals

5 types of literals

1. Integeral Literal:
byte b = 100;
short b1 = 200;
int b2 = 400;
long b3 = 40000;

2. Floating Literals
float f1 = 23.34f;
float f2 = 23.34;

3. Boolean literal
boolean b1 = false;

4.Char literal:
char c = 'a';

5.String literal
String s = "yaane";

We cannot write more than one character with in the single quote.

Ex: 'a' or '4'

'abc' //wrong

boolean literal it is always either true or false.

String literal always within the double quotes.

float literal always ended with either small 'f' or 'F'

Ex: float f1 = 23.23f;

float f2 = 56.61F;

ended 'f' or 'F' is mandatory.

double d = 34.45;

double d1 = 34.45d;

double d2 = 34.45D;

double literal can be followed by either 'd' or 'D'. It is not a mandatory.

By default all number in java comes under int literal.

By default all fractional numbers in java comes under double literal.

```
long l = 100;
```

```
long l1= 100L;
```

```
long l2 = 100l;
```

Ended 'l' and 'L' is not mandatory.

Keywords:

Keywords are predefined words or reserved words.

These are having some functionality through logic.

Keywords are used to decrease the size of a program.

With help of keywords we can develop the program within the less time.

These functionalities will help us to communicate with both compiler and JVM.

Whenever our program executes internally keywords functionalities also executed.

We have 50 keywords in java among them 48 are used and 2 are unused

false, true, null these are also predefine words but these not comes under keywords, these are comes under literals.

```
boolean b = false;
```

```
boolean b1 = true;
```

```
String s = null;
```

List of keywords:

java files:

class
interface
enum

data types:

byte
short
int
long
float
double
char
boolean

For return type:

void.

Controle statements:

if
else
switch
case
default

Looping statements:

for
do

while

Transfer the control:

break

continue

return

AccessModifer:

private

public

protected

Modifer:

final

abstract

synchronized

strictfp

volatile

native

transient

object related keyword:

this

super

instanceof

relationship:

extends

implements

package:

package

import

Exception handling:

try

catch

finally

throw

throws

assert

memory allocation:

static

new

unused:

goto

const

Note: keywords must be writes in small case.

String is a predefine class, which is available in java.lang package. But we can use String as a data type.

50 (48--used)	Keywords =====	2--unused (goto,const)
1.class	13.static	23.break 32.extends
2.interface	14.new	24.continue 33.implements
3.enum		25.return
	15.if	
4.byte	16.else	26.this 34.package 42.final
5.short		27.super 35.import 43.abstract
6.int	17.for	28.instanceof 36.try 44.synchronized
7.long	18.while	
8.float	19.do	29.private 37.catch 45.transient
9.double		30.protected 38.finally 46.strictfp
10.char	20.switch	31.public 39.throw 47.native
11.boolean	21.case	
12.void	22.default	40.throws 41.assert 48.volatle

1.

Separtors:

It is a small java element which is used to differentiate from one java code to another code (element).

These are some of the special characters.

Ex: We have totally eight separators.

Separator:

```

int a;
int a[];                                <>

int age;
int age();                                sop("hi");
int age{};                                sop("bye");

class A{};                                switch(-){}

class B{                                   case 1:
}                                         case 2:
}                                         {1, 2, 3, 4, 5}

class to method  System.exit(0)
System.out.print

```

Those are:

"()" To differentiate variable to method

"{}" Differentiate from class to method or method to method (scope).

"[]" Differentiate primitive variable to reference variable.

- “.” Make a relation between class to method and class to variable
- “,” Use to separate one value to another value
- “;” To separate one statement to another
- “:” To separate one case to another in switch
- “<>” To separate normal collection object to generic

Escape Characters:

These are used print the data in different locations/format. (Style).

```
class EscapeCharacters{
    public static void main(String...ram){
        //System.out.println("hi\tbye");
        //System.out.println("hi\nbye");
        //System.out.println("hi\bbye");
        //System.out.println("hi\rbye");
        System.out.println("hi\'bye");
        System.out.println("hi\"bye");
        System.out.println("hi\fbye");
    }
}
```

Data types:

Data types are pre-reserved words, which is used to specify the type of literal/value.

```
int a = 10;
boolean b = false;
```

With help of data type we can provide the memory for variables.

```
int a = 10; // 4 bytes of memory.
```

Data types are allows valid operations.

```
boolean b = true;  
b = b++; //invalid syntax
```

Data types are provides proper result/output.

```
String s1 = "10";  
String s2 = "20";  
S.o.p(s1+s2); //1020  
int a = 10;  
int b = 20;  
S.o.p(a+b); //30
```

Data types are used to differentiate the one value to another value.

Data types are provides proper memory management.

Classification of data types:

Diagram relation to classification.

Q) Why java uses these many data types?

A) To use the memory in proper manner then java uses different data types.

If the value is small then we can go for small range of data type.

If the value is big then we can go for higher range of data type.

Ranges of data types:

byte: (1 byte)---> $1 * 8 = 8$ bits

byte b = 10;

If we want to place the data in the memory, first we should convert into binary number system. (0,1). Zero and one are two digits.

8

$2^7 = 256$ 7
--- = 128 = 2

2

-ve number, + numbers

-128 to 128

-128 to 0 to 127

7 7
-2 to 0 to 2 -1

8-1 8-1
-2 to 0 to 2 -1

The range of float, double is greater than the byte,short,int,long datatypes.

Casting:

Converting from one data type variable value to another data type variable value is called Casting.

Casting is two types.

1. Implicit casting.
2. Explicit casting.

1. Implicit Casting: Converting lower data type value to higher data type values is called implicit casting.

We can pass the values to variables within the range only we cannot pass beyond the range values.

```
int a = 10;
```

- 1) In the above statement compiler will check the destination variable type (int).
- 2) Based on the type, compiler will check range
- 3) Compiler is also checking the value of a variable(source value).
- 4) If value is within the range then compiler will not give any error otherwise compiler will give one compiler time error. That is possible lossy conversion.

Sometimes it will give incompatible error.

```
class Casting{  
    public static void main(String args[]){  
        System.out.println("main method");  
        byte b = 127;  
        System.out.println("b: "+b);  
        byte b2 = -128;
```

```
        System.out.println("b2: "+b2);
        //byte b1 = 128;
        //System.out.println("b1: "+b1);
        short s = 32767;
        System.out.println("s: "+s);
        short s1 = -32769;
        System.out.println("s1: "+s1);
    }
}
```

```
class Casting{
    public static void main(String... ram){
        char b8 = 'A';
        int b9 = b8;
        System.out.println(b9);
        byte b1 = 10;
        short b2 = b1;
        int b3 = b1;
        System.out.println(b3);
        float b4 = b1;
        System.out.println(b4);
        long b5 = 123L;
        float b6 = b5;
        System.out.println(b6);
    }
}
```

```

float b7 = 34.34F;//float must be ended           //with either 'f' or 'F'
char b10 = 97;
System.out.println(b10);
for(int i=48;i<58;i++){
    System.out.print((char)i+" ");
}
System.out.println();
for(int i=65;i<=90;i++){
    System.out.print((char)i+" ");
}
System.out.println();
for(int i=97;i<=122;i++){
    System.out.print((char)i+" ");
}
System.out.println();
char c = '?';
System.out.println((int)c);
}
}

```

Assigning the values to variables:

We have 7 ways to assign the values to variables.

1. Assigns the values directly to variable.

```
int a = 111;
```

2. Assign the values to variable by using another variables.

```
byte b = 222
```

```
int c = b;
```

3. Assign the values to variables by using method return type.

```
int m1          =  m2(); //calling area
```

Destination part initialization part

```
int m2(){ //called area
```

```
    return 333;
```

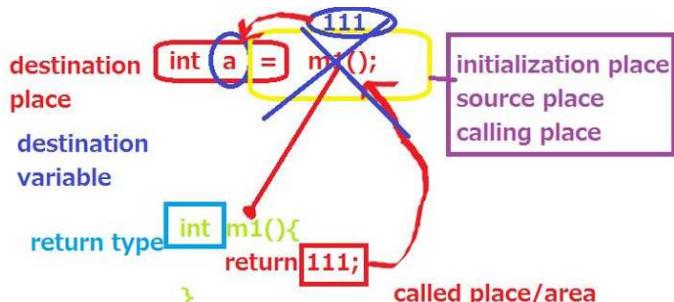
```
}
```

Note: "return" statement will forward the data from called area to calling area.

If any method calling is available in initialization part, then that method must be non-void method.

non-void method means, method carrying some information. That information must be compatible with destination variable datatype.

Method must have return type. Return type also must be compatible with destination variable datatype. If any method having return type then that method must have return statement with value. Again value must be compatible with method return type.



with the help of method return type we can recognize, what type of information is carrying by the method.

return statement, we can forward the information from called area to calling area.

Internal functionalities of compiler meanwhile of assiging the value to variable with the help of method return type.

```
byte b = m1();
1. compiler check destination variable type(byte)
2. compiler check destination variable type range(-128 to 0 127)
3. compiler check source value( m1())
4. compiler controle goes to method(m1()) and check method return
   type and return value compatiable or not
   if not compiler error
   if yes
      compiler checks method return type and destination
      variable type compatiable or not
      if not compiler time error
      if yes no error.
```

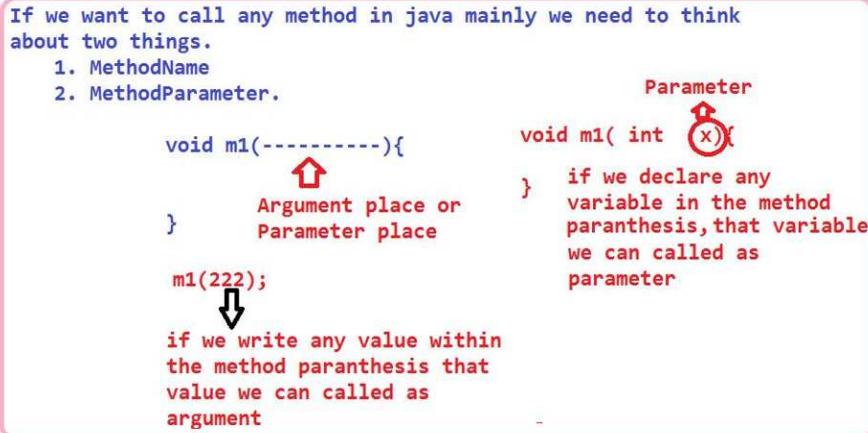
4. Passing the value as a argument to other method.

```
void m3(String s){
}
```

calling the m3 method: m3("ram");

```
void m2(boolean b){
}
```

calling the m2 method: m2(true);



5) We can assign the value to variable with the help of expression.

`int c = 10+20; //here 10+20 is an expression`

`int a = 10;`

`int b = 20;`

`int d = a+b; //a+b is an expression.`

6) By using cast operator.

`byte b = (byte)130;`

7) By using object.

`int I = new Integer(100);`

Functionalaties of compiler and jvm in the casting:

compiler:

compiler checks type of an variables(both source and destination variables)

Based on the type, compiler checks range of an variabels.(datatype).

If Destination variable Range greater than source variables Range then compiler will satisfy otherwise compiler will give error.

(DVR>=SVR)

Compiler not check any value.

```
short b = 111;  
int i = b;//valid  
DVR >= SVR
```

```
byte b1 = b;//not valid
```

Jvm: jvm will check type of an variable, range of an variable, DVR \geq SVR or not, finally jvm will check "value" also.

Note: if we assign the value directly to variable compiler will check type and range and value.

```
byte b = 127;
```

ExplcitCasting: Converting from higher datatype value to lower datatype value.

By default it is not possible in java.

but we can do with the help of cast operator.

```
int b = 127;  
byte b1 = b;//invalid  
byte b2 = (byte)b;//valid
```

In the explicity casting, if the source value is with in the range, then jvm will place the same value assign to destination variable. otherwise jvm will place some other value.

```
public class ExplacitCasting {  
  
    public static void main(String[] args) {  
        byte b = 127;  
        int i = b;  
  
        int j = 127;  
        byte b1 = (byte)j;  
        System.out.println(b1);  
  
        int k = 130;  
        byte b2 = (byte)k;  
        System.out.println(b2);  
  
        int k1 = 150;  
        byte b3 = (byte)k1;  
        System.out.println(b3);  
  
        int k2 = 300;  
        byte b4 = (byte)k2;  
        System.out.println(b4);  
    }  
}
```

Note: we can not convert boolean data type values to another data type.

```
boolean b = false;
```

```
int a = b;//invalid
```

boolean value we cannot placed into other datatype and any other datatype value cannot placed into boolean datatype.

```
byte b = 100;
```

```
boolean b1 = b;//invalid
```

```
*****
```

```
public class ExplacitCasting {
```

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

```
        char a = 'a';
        System.out.println();
        System.out.println("=====");
        System.out.println(a);//s.o.p('a');
        char a1 = 'z';
        System.out.println(a1);
        //System.out.println(ram); //invalid
        //ram is a variable we didn't declare
        System.out.println("a");
        System.out.println('a');
        //System.out.println('ab');//invalid
        System.out.println("ram");
        char a2 = 100;
        System.out.println("==========");
        System.out.println(a2);
        //char a3 = -100;//invalid
        char a4 = 0b1000001;
        System.out.println(a4);
        char a5 = 00061;
        System.out.println(a5);
        char a6 = 0x0061;
        System.out.println(a6);
        char a7 = '\u0062';
        System.out.println(a7);
        System.out.println("unicode values for A to Z");
        for(int i=65;i<=90;i=i+1){
            System.out.println(i+"...."+(char)i);
    }
```

```
System.out.println("unicode values for a to z");
for(int i=97;i<=122;i=i+1){
    System.out.println(i+"...."+(char)i);
}
System.out.println("unicode values for numerics");
for(int i=35;i<=64;i=i+1){
    System.out.println(i+"...."+(char)i);
}
char a8 = 100;
//byte a9 = a8;//invalid
//short a10 = a8;//invalid
char a11 = 0;
//boolean a12 = a11;//invalid

char c1 = 'a';

//char c2 = 'ab';

//we cannot write more than one character
//with in the single quote

char c2 = 100;

System.out.println(c2);

System.out.println((int)' ');

System.out.println((int)'f');

char c3 = '\u0061';//hexadecimal           //number system

System.out.println(c3);

char c4 = 0061;//octal number system

System.out.println(c4);

//short s = c4;

//boolean c5 = false;

//byte b = c5;

//boolean c6 = c4;
```

```
float f = 23.34f;  
int f1 = (int)f;  
System.out.println(f1);  
  
double f2 = 435.1234567890d;  
long f3 = (long)f2;  
System.out.println(f3);  
float f4 = (float)f2;  
System.out.println(f4);  
}  
}
```

char variable can be hold character values, numerical values, binary, octal, hexadecimal, Unicode type values.

Difference between print() and println():

with the help of print() and println(), we can print the data on the console.

print() is printing on the console, after printing the data the cursor position is in the same line, whereas println() will print the data on the console, after printing the data the cursor position is in the next line.

ex: s.o.print("hi");

s.o.print("bye");

o/p: hibye

ex: s.o.println("hi");

```
s.o.println("bye");
```

o/p: hi

bye

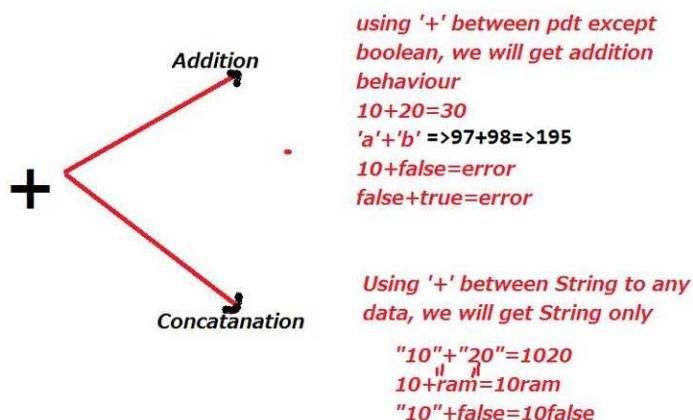
In java we have only "+" operator is overloaded.

'+' is used for both

- 1.concatanation
2. addtion

if we are using '+' in between two number datatypes then we will get addition functionalaties.

if we are using '+' in between number and string then we will get concatanaation functionalaties.



within the `println()` statement we can write

- 1)empty.
- 2)value
- 3)variable
- 4)expression

5)non-void method calling statement

6)reference/object

7)String

8)array

```
public class ExplacitCasting {  
    public static void main(String[] args) {  
        double d = 23.34D;  
        float f = (byte)(char)(int)(short)(int)(double) (long)(float)d;  
        byte b1 = 100;  
        byte b2 = 20;  
        //byte b3 = b1+b2;  
        //byte+byte=int.  
        int a = 111;  
        int b = 222;  
        int c = a+b;  
        //int+int = int;  
        byte b3 = 100;  
        int b4 = 100;  
        //short b5 = b3+b4;  
        //byte+int=int  
        short b6 = 100;  
        short b7 = 200;  
        //short b8 = b6+b7;
```

```
//short + short = int
//short b9 = b3+b7;
//byte+short=int

System.out.println("10+20: "+(10+20));

int f1 = 111;
int f2 = 222;

System.out.println("f1"+"+"+"f2:"+" " + (f1+f2));

System.out.println(11+"ram");
System.out.println("ram"+22);
System.out.println("ram"+"22");
System.out.println(10+20+"ram");

System.out.println(10+20+"ram"+10+20);
System.out.println(10+20+"ram"+(10+20));

//System.out.println(10+false);

System.out.println("ram"+'A');
System.out.println('A'+32);

//int+char = int

System.out.println(32+'A');

long ll = 34l;
float ff = ll;
long l2 = ll+ff;

//long+float=float

}

}
```

Comments:

If we want write any statements not related to java grammar rule then we can write within the comments.

Comments are used to provide the information about java elements.

What ever the code is available under the comments, that code simply ignored by the compiler that comments source code not converted into byte code. So JVM is also ignored.

With the help of the comments we can easily understand program.

Java provides three types of comments

1. Single line comment. (//)
2. Multiline comment. (/*.....*/)
3. Document comment.(/**.....*/)

Single Line Comment:

By using this comment we can write only one line of code.

This can be represent with "://" (two forward slashes)

Multiline comments:

By using this comment we can comment more than one line of code.

This can be represents with "/*.....*/".

Document comment:

This is also same as multiline comment; By using this we can comment more than one line of code.

This is represents with "/***/"

Example:

```
/*Author: Ram  
DOW:14/12/2015  
Vendor: nit  
java Version: 1.7  
*/  
  
public class CommentDemo {  
    //Without main method we can't execute the //program.  
    public static void main(String[] args) {  
        /**  
         * int a = 10;  
         *     int b = 20;  
         *System.out.println(a+b);  
        */  
    }  
}
```

Valid comments:

```
// // / / /  
// /* */  
// /** */  
/* // */  
/* /* */  
/** /** */
```

```
/** /* */
```

Invalid comments:

```
/*  */ */  
/**/** */  
/**/* */  
/** /* */ */  
/* /** */ */
```

Escape characters:

```
class EscapeDemo{  
    static public void main(String...ram){  
        //System.out.println("hibye");//hibye  
        //System.out.println("hi\nbye");      //hi  
                                         //bye  
        //System.out.println("hi\tbye");//hi     bye  
        //System.out.println("hi\bbye");//hbye  
        //System.out.println("hi\bbye");//error  
        //System.out.println("hihi\bbye");//hihbye  
        //System.out.println("hi\rbye");//bye  
        //System.out.println("hihi\rbye");//byei  
        //System.out.println("hi\fbye");  
        //System.out.println("hi\b'ye");//hi'bye  
        System.out.println("hi\"bye");//hi"bye
```

```
 }  
}
```

Note: Don't give any space between '\' and character (t, n, b, r, f, ', ", \').

Variable:

Variable is a named memory location, it can be holds the value, the value can be change from one statement to another statement.

```
int a = 10;//value --10  
a = a+1; //value--11  
a = a*2;//value--22.
```

Based on the data types variables can be classified into two types.

1. Primitive Variable
2. Referenced Variable

Primitive Variable:

A variable, which can define with the help of primitive data type is called primitive variable.

```
Ex: int x=10;  
float y=45.56f;
```

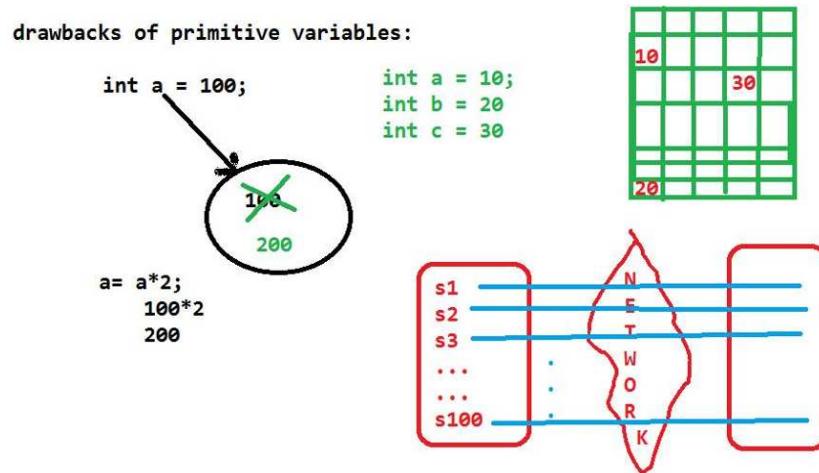
Drawbacks of Primitive Variable:

1. Primitive variables can hold only single value at a time.
It is unable hold more than one value.

2. If we are placing the data into memory with the help of primitive variables, the data is not stored in the memory in sequential order, the data will be stored in random order.

While retrieving the data from memory will take more time.

3. If we are sending the information from machine to another machine with the help of primitive variable, will take more network calls.



If we want resolve above draw backs then we can go for referenced data types.

1. Array
2. Class
3. Interface
4. Enum
5. Annotation

Array: Array is a single entity which holds continuous memory locations, which hold more than one value with same data type or its lower range data types.

```
int a[] = {10,20,30};
```

Each and every element can recognize, with the help of index position.

Within the one index position, we can write only one value.

Q) How to represents arrays in java?

Holding multiple values

We need "{}"

To differentiate one value to another we need ','

Statements ended with semicolon';'

For assignment we need '='

To hold value we need Variable

To represents array '[]'

To represent type of values data type

```
int[]a = {10, 20, 30, 40, 50};
```

```
array:  
-->is a single entity  
-->contains mutiple memory locations for  
-->holding multiple values of  
-->same type(homogeneous)/lower range type/  
type(hetrogeneous).  
-->in a sequence order.
```

Q)What is the use of index?

A) it is used to read the data or write the data from or into array.

Q) Why should we use index?

A) We can not determin the memory locations/address of values.

The memory locations are changed from one time execution to another time.

To resolve these problem we should we use index.

How to represents arrays in java?

- >holding multiple values
- > we need "{}"
- >to differentiate one value to another we need ','
- >statements ended with semicolon ;'
- >for assignment we need '='
- >variable
- >to represents array '[]'
- >to represent type of values datatype

```
int[]a = {10, 20, 30, 40, 50};
```

array: collection of memory blocks

multiple values
of same type / memory
placed into single entity

represents---> "[]"

a

index

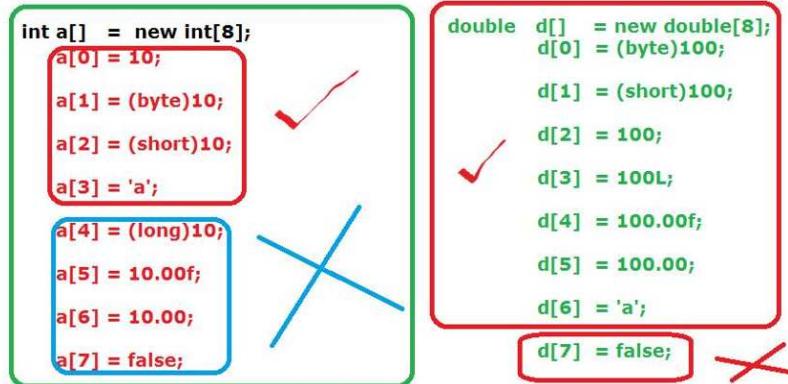
0	1	2	3
10	20	30	40

cell/memory value/information/
data/element

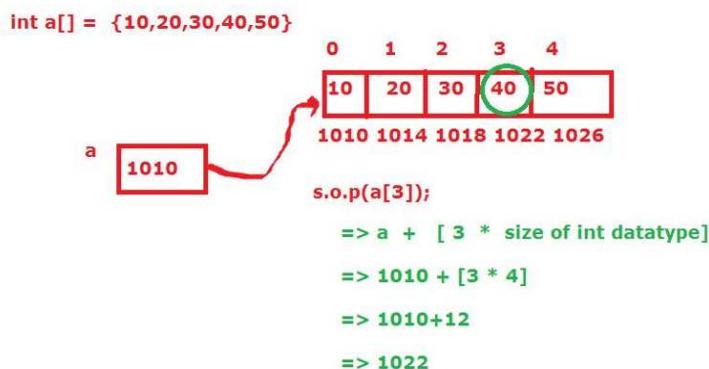
$a[0]=10$
 $a[2]=30$
 $a[1]=20$
 $a[3]=40$

length field vs length():

array is a single entity, which contains multiple memory location to hold multiple values of same/similar/homogeneous data. or it lower types.



Note: With the help of cast operator we can do both implicit and explicit casting.



Representation of arrays:

We have three ways to represent arrays.

1. int a[] = {11,22,33,44};
2. int b[] = new int[5];
//here 5 is called size or dimensions
//mention the size is mandatory.
3. int c[] = new int[]{11,22,33,44,55};

Arrays can be represented with help of "[]".

Difference between length field and length ():

Length filed can be applicable on array variables, to know the size of an array.

```
int a [] = {0,1,2,3,4,5,6,7,8,9};  
s.o.p(a.length());//10  
  
String s[] = {"kiran","ram","suji","varun","kishore"};  
s.o.p(s.length()); //5  
s.o.p(s[4].length()); //7
```

length (): This is used for to know the number character are existed in string.

```
String s = "internationalization";  
s.o.p(s.length());//20  
  
String s1 ="ram chandra rao";  
s.o.p(s1.length()); //15
```

Arrays can be classified in the following manner.

1. Single dimensional array
2. Double dimensional array
3. Multi dimensional array

Single dimensional array:

Store the elements either in horizontal (rows) or vertical manner (columns).

Ex: int a[] = {111,222,333,444,555};

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

```

int a[] = {111,222,333,444,555};

/*System.out.println(a[0]);
System.out.println(a[1]);
System.out.println(a[2]);
System.out.println(a[3]);
System.out.println(a[4]);*/
for(int i = 0; i < a.length; i = i+1 ){

    System.out.println(a[i]);
}

}

```

Randomly selecting the data from array by using Random class:

```

import java.util.Random;
public class ArrayDemo {
    public static void main(String[] args) {
        int a[] = {11,22,33,44,55};
        Random r = new Random();
        int count=0;
        while(count<a.length){
            for(int i=0;i<a.length;i++){
                int rsn = r.nextInt(60);
                if(rsn==a[i]){
                    System.out.println("rsn: "+rsn);
                    count++;
                    break;
                }
            }
        }
    }

package array;

import java.util.Scanner;

```

```

public class ArrayDemo {
    public static void main(String[] args) {
        /*System.out.println("main method");
        int[] a = {11,22,33,44,55};
        System.out.println("=====for loop=====");
        for(int i=0;i<a.length;i++){
            System.out.println(a[i]);
        }
        System.out.println("=====while loop=====");
        int j=0;
        while(j<a.length){
            System.out.println(a[j]);
            ++j;
        }
        System.out.println("=====do-while loop=====");
        int k=0;
        do{
            System.out.println(a[k]);
            ++k;
        }while(k<a.length);
        System.out.println("=====for-each loop=====");
        for(int a1 : a){
            System.out.println(a1);
        }*/
        /*int[] a = {11,22,33,44,55};
        System.out.println("=====for loop=====");
        for(int i=0;i<a.length;i++){
            System.out.println(a[i]);
        }
        System.out.println("enter to change the values");
        for(int i=0;i<a.length;i++){
            System.out.println(a[i]);
            a[i]=new Scanner(System.in).nextInt();
        }
        System.out.println("=====for loop=====");
        for(int i=0;i<a.length;i++){
            System.out.println(a[i]);
        }*/
        int[] a = {11,22,33,44,55};
        System.out.println("=====for-each loop=====");
        for(int a1 : a){
    
```

```

        System.out.println(a1);
    }

System.out.println("enter the data for change");
for(int a1 : a){
    System.out.println(a1);
    a1=new Scanner(System.in).nextInt();
}
System.out.println("=====for-each loop=====");
for(int a1 : a){
    System.out.println(a1);
}
}
}

```

Java provides 6 ways to read the data from keyboard.

1. Command Line Arguments
2. java.util.Scanner
3. java.io.BufferedReader
4. java.io.Console.
5. java.io.DataInputStream
6. GUI (awt,swing,applet).
7. System Properties

Command Line Arguments:

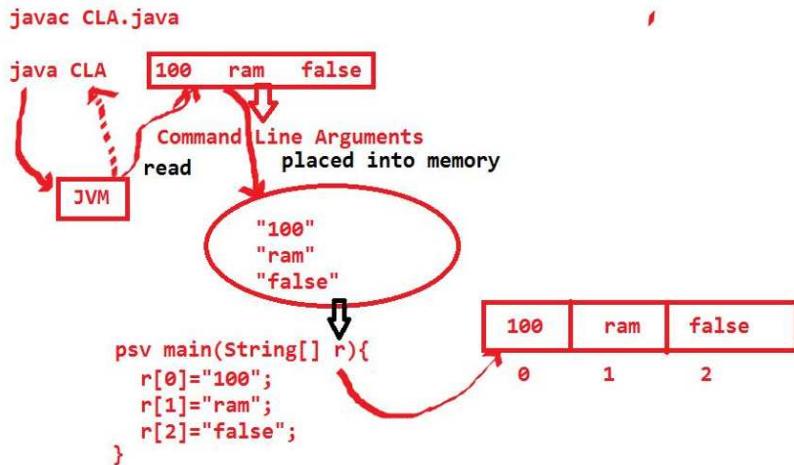
Passing values at mean while executing the program is called command line argument.

Syntax: java ArrayDemo 111 ram (click on enter button)

Here 111, ram are called command line arguments.

Whatever the data, which is sending from keyboard to java application, that data is always in the form of string.

The data may numerical ("123"), character ("net"), special character ("%^&*"), alphanumerical ("ram124");



```

class CommandLine{

    static public void main(String[] ram){

        String s1 = ram[0];
        String s2 = ram[1];
        System.out.println(s1);
        System.out.println(s2);
        System.out.println("s1+s2: "+(s1+s2));
    }
}
  
```

```

javac CommandLine.java
java CommandLine php android
php
android
s1+s2: phpandroid
  
```

```

java CommandLine 123 234
  
```

```
123  
234  
s1+s2: 123234
```

```
java CommandLine "#$%^" "&*("
```

```
#$%^  
&*()  
s1+s2: #$%^&*
```

Invalid Execution:

```
java CommandLine (enter)  
java.lang.ArrayIndexOutOfBoundsException: 0  
java CommandLine 123 (enter)  
java.lang.ArrayIndexOutOfBoundsException: 1  
class CommandLine{  
    static public void main(String[] ram){  
        String s1 = ram[0];  
        System.out.println(s1);  
        String s2 = ram[1];  
        System.out.println(s2);  
        System.out.println(s1+s2);  
        int i = Integer.parseInt(s1);  
        System.out.println(i);  
        int j = Integer.parseInt(s2);  
        System.out.println(j);
```

```
        System.out.println(i+j);  
    }  
}
```

The drawback of above program is , we read the data in the form of string, later we will converting into respective data type(int, boolean, float).

So here we are writing some extra conversion logic to read the data in different format.

```
class CLA{  
    public static void main(String[] s){  
        String s1= s[0];  
        boolean b1 = Boolean.parseBoolean(s1);  
        System.out.println(b1);  
  
    }  
}
```

If we are sending otherthan "true" value from command line argument, we are always getting like false.But jvm never giving any proper Exception.

We need to remember the number(count) of data.

We need to remember the order(type) of data.

Based on order we need enter the data in keyboard.

To avoiding this problem, we should go for java.util.Scanner class.

```
class CommandLineArgument{  
    public static void main(String[] s){  
        System.out.println("main method");  
        String s1 = s[0];
```

```

        boolean b = Boolean.parseBoolean(s1);

        System.out.println(b);

    }

}

```

If we are sending other than true value, we are always getting false only,
but JVM never giving exception message like
java.lang.NumberFormatException

```

public class Account {
    long accno;
    String accHolderName;
    short pin;
    double amount;
    public static void main(String[] args)
    throws IOException{
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        Account[] acc = new Account[2];
        for(int i=0;i<2;i++){
            Account cust1 = new Account();
            acc[i] = cust1;
            System.out.println("enter accno of long type");
            String s = br.readLine();
            cust1.accno = Long.parseLong(s);
            System.out.println("enter accHolderName of String type ");
            cust1.accHolderName = br.readLine();
            System.out.println("enter pin number of short type");
            String s1 = br.readLine();
            cust1.pin = Short.parseShort(s1);
            System.out.println("enter amount of double type");
            cust1.amount = Double.parseDouble(br.readLine());

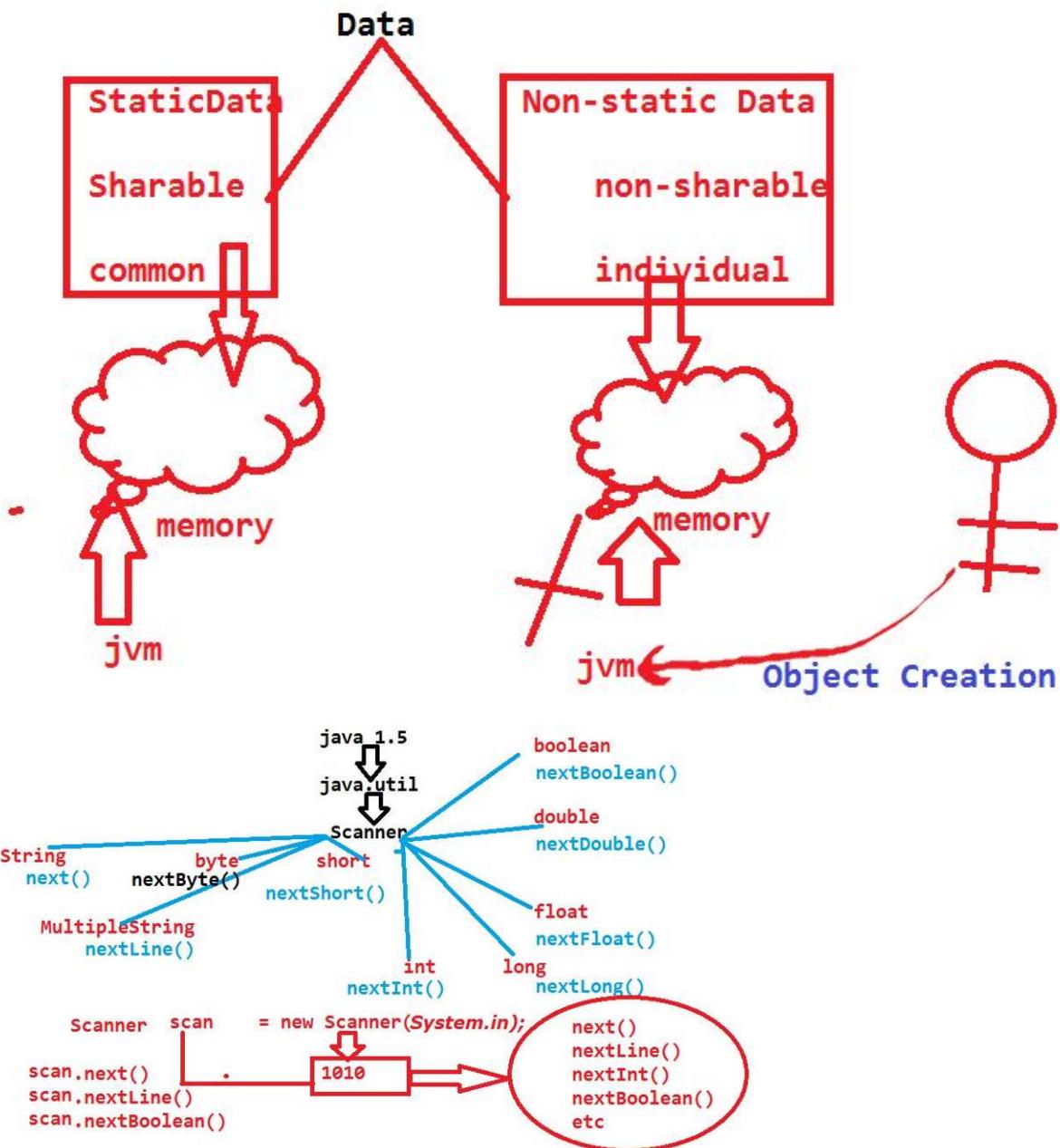
            System.out.println(cust1.accno);
            System.out.println(cust1.accHolderName);
            System.out.println(cust1.pin);
            System.out.println(cust1.amount);
            System.out.println("=====");
        }
        System.out.println(acc[0].accHolderName);
        System.out.println(acc[1].accHolderName);
    }
}
=====
```

java.util.Scanner:

Scanner is an one predefine class, which is used to read the data in the different data format.

Different data format means either String, int, byte, short, long, float, double, boolean.

With the help of Scanner class we cannot read character data.



```
import java.util.Scanner;

public class ScannerDemo {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        System.out.println("enter some string value");

        String s = scan.nextLine();

        System.out.println(s);

        System.out.println("enter some string value");

        String s1 = scan.next();

        System.out.println(s1);

        System.out.println("enter some integer data");

        int i = scan.nextInt();

        System.out.println(i);

        System.out.println("enter some boolean data");

        boolean b = scan.nextBoolean();

        System.out.println(b);

        System.out.println("enter some float data");

        float f = scan.nextFloat();

        System.out.println(f);

    }

}
```

```
import java.util.Scanner;

public class ScannerDemo {
```

```
public static void main(String[] args) {  
    Scanner scan = new Scanner(System.in);  
    System.out.println("enter some string value");  
    String s1 = scan.next();  
    System.out.println(s1);  
    scan.nextLine();  
    System.out.println("enter some string value");  
    String s = scan.nextLine();  
    System.out.println(s);  
}  
}
```

Note: In the above program, if we are not

Using obj.nextLine () syntax, then we are unable to read more words by using second nextLine() method. The reason is next () will read only collection of characters after that if we click enter button, that enter button value will be read by nextLine ().

So we are unable type some words.

To overcome the above problem we need to use one extra nextLine() syntax in our program.

```
import java.util.Scanner;  
public class ArrayDemo1 {  
    public static void main(String[] args) {  
        int a[] = new int[5];  
        Scanner scan = new Scanner (System.in);  
        System.out.println("Reading the valued from keyboard");  
        for(int i = 0; i < a.length; i = i+1){
```

```
a[i] = scan.nextInt();

}

System.out.println("printing the values of a array");

for(int i = 0; i< a.length; i++){
    System.out.println(a[i]);
}

}

}

// Copy the elements from one array to another

public class ArrayDemo1 {

    public static void main(String[] args) {

        int a [] = {11,22,33,44,55};

        int b [] = new int[5];

        for(int i =0; i < a.length ; i++){

            //reading the elements from a array

            //b[i] = a[i];

            for(int j =i; j<i+1; j++ ){

                //placing elements into b array

                b[j] = a[i];
            }
        }

        System.out.println("printing b array elements");
    }
}
```

```

        for(int i =0;i<b.length;i++){
            System.out.println(b[i]);
        }
    }
}

```

Whenever we read float data and double data by Scanner class don't send float and double followed by either 'f' or 'F' and 'd' or 'D'.

```

import java.util.Scanner;
public class ScannerDemo {
    public static void main(String[] args) {

        /*long l1 = 100;
        long l2 = 100l;
        long l3 = 100L;

        //float f1 = 123.456;
        float f2 = 100;
        float f3 = 345.45f;
        float f4 = 345.45F;

        double d1 = 400;
        double d2 = 12.23;
        double d3 = 12.23d;
        double d4 = 12.23D;*/

        Scanner scan = new Scanner(System.in);
        System.out.println("enter some float data");
        float f1 = scan.nextFloat();
        System.out.println("f1: "+f1);

        System.out.println("enter some double data");
        double d1 = scan.nextDouble();
        System.out.println("d1: "+d1);
    }
}

```

Java provides one predefine method for copy the elements from one array to another array.

That is `arrayCopy()`. It is static method, which is available at `java.lang.System`.

```
System.arraycopy(a,0,b,0,5);
```

a--> source array.

0-->from which index position we are reading the elements from source array.

b--> destination array.

0--> In which index position, we are going to be place the elements in destination array

5-->Number of elements.

Double Dimensional Array:

```
import java.util.Scanner;

public class ArrayDemo1 {

    public static void main(String[] args) {

        int a[][] = {{11,22,33},{44,55,66}};

        for(int i =0;i<2 ; i++){//rows

            for(int j=0;j<3;j++){//columns

                System.out.print(a[i][j]+" ");

            }

            System.out.println();

        }

    }

}

import java.util.Scanner;

public class ArrayDemo1 {

    public static void main(String[] args) {

        int a[][] = new int[3][3];

        System.out.println("enter some data for 3*3 matrix");

    }

}
```

```

Scanner scan = new Scanner(System.in);

for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
        a[i][j] = scan.nextInt();
    }
}

System.out.println("data from a array");

for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
        System.out.print(a[i][j]+" ");
    }
    System.out.println();
}
}
}

```

Difference between char array to remaining data type arrays:

If we are printing char array reference variable we will get data. If we are print remaining data type array variables we will reference.
(classname@memory)

```

public class ArrayDemo {

    public static void main(String[] ram) {
        char c[]={1,'a','o'};
        System.out.println(c);
    }
}

```

```
int a[] = {11,22,33,444};  
System.out.println(a);  
boolean b[] = {false,true,true,false};  
System.out.println(b);  
}  
}  
  
public class ArrayDemo {  
public static void main(String[] ram) {  
    byte b[] = new byte[2];  
    System.out.println(b.getClass().getName());  
    short s[] = new short[3];  
    System.out.println(s.getClass().getName());  
    int i[] = new int[2];  
    System.out.println(i.getClass().getName());  
    float f[] = new float[3];  
    System.out.println(f.getClass().getName());  
    double d[] = new double[7];  
    System.out.println(d.getClass().getName());  
    char c[] = new char[1];  
    System.out.println(c.getClass().getName());  
    System.out.println("*****");  
    long l[] = new long[2];  
    System.out.println(l.getClass().getName());  
    boolean b1[] = new boolean[2];
```

```
        System.out.println(b1.getClass().getName());  
    }  
}
```

Anonymous arrays:

An array, which does not have any external reference, that array is called anonymous array.

Ex: new int []{11,12,13,14,15};

With the help of anonymous array we cannot reuse the memory.

With the help of anonymous array we can interact with the data only one time.

In different times we can send different number of values and different values. After using memories GC will clean them.

```
public class ArrayDemo {  
  
    static void m1(int x[]){  
  
        for(int i=0;i<x.length;i++){  
  
            System.out.println(x[i]);  
  
        }  
  
        System.out.println("*****");  
  
    }  
  
    public static void main(String[] ram) {  
  
        //reference array  
  
        int a[] = new int []{11,22,33,44};  
  
        m1 (a);  
  
        //anonymous array  
  
        m1 (new int []{111,222,333,444});  
    }  
}
```

```
 }  
}
```

Multidimensional Array:

```
import java.util.Scanner;  
  
public class ArrayDemo {  
  
    public static void main(String[] ram) {  
  
        int a[][][] = new int[2][2][3];  
  
        Scanner scan = new Scanner(System.in);  
  
        System.out.println("please enter integer" +  
                           " values for 2*2*3 matrix");  
  
        for(int i=0;i<2;i++){  
  
            for(int j=0;j<2;j++){  
  
                for(int k=0;k<3;k++){  
  
                    a[i][j][k]= scan.nextInt();  
  
                }  
  
            }  
  
        }  
  
        System.out.println("elements of a array");  
  
        for(int i=0;i<2;i++){  
  
            for(int j=0;j<2;j++){  
  
                for(int k=0;k<3;k++){  
  
                    System.out.print(a[i][j][k] + " ");  
  
                }  
  
            }  
  
            System.out.println();  
        }  
    }  
}
```

```
    }  
    System.out.println();  
}  
}  
}
```

Jagged array:

If we want to save the memory while inserting the data into the array, we can go for jagged array.

```
Scanner scan = new Scanner (System.in);  
  
int a[][] = new int[4][];  
  
a[0] = new int[1];  
  
a[1] = new int[2];  
  
a[2] = new int[3];  
  
a[3] = new int[4];  
  
System.out.println("enter elements for jagged array");  
  
for(int i=0;i<4;i++){//rows  
  
    for(int j=0;j<i+1;j++){  
  
        a[i][j] = scan.nextInt();  
  
    }  
}  
  
System.out.println("a array elements");  
  
for (int i=0;i<4;i++){//rows  
  
    for(int j=0;j<i+1;j++){  
  
        System.out.print(a[i][j]+ " ");  
    }  
}
```

```
    }System.out.println();  
  
}  
  
import java.util.Scanner;  
  
public class ScannerDemo {  
    public static void main(String[] args) {  
        int[][][] a= new int[2][4][];  
        for(int i=0;i<2;i++){  
            a[i][0] = new int[1];  
            a[i][1] = new int[2];  
            a[i][2] = new int[3];  
            a[i][3] = new int[4];  
        }  
        Scanner scan = new Scanner(System.in);  
        System.out.println("enter some data for jagged array");  
        for(int k=0;k<2;k++){  
            for(int i=0;i<4;i++){  
                for(int j=0;j<=i;j++){  
                    a[k][i][j]= scan.nextInt();  
                }  
            }  
        }  
        System.out.println(" jagged array data:");  
        for(int k=0;k<2;k++){
```

```

        for(int i=0;i<4;i++){
            for(int j=0;j<=i;j++){
                System.out.print(a[k][i][j]+" ");
            }
            System.out.println();
        }
        System.out.println();
    }

}

import java.util.Scanner;

public class Demo {
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("enter row size");
        int row = scan.nextInt();
        int[][] a = new int[row][];
        int []cc = new int[row];
        for(int i=0;i<row;i++){
            System.out.println("how many elements do you need in row-"+(i+1));
            int column = scan.nextInt();
            cc[i] = column;
            a[i] =new int[column];
            System.out.println("enter values");
            for(int j=0;j<column;j++){
                a[i][j] = scan.nextInt();
            }
        }
        for(int i=0;i<row;i++){
            for(int j=0;j<cc[i];j++){
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
    }
}

```

Array with casting values:

Array can hold different type of data, but the data must be in the range of array type.

```
public class ArrayDemo {  
    public static void main(String[] args) {  
        int a[] = new int[5];  
        a[0]=10;  
        a[1]=(byte)120;  
        a[2]=(short)250;  
        a[3]='a';  
        //a[4]=(float)120.35;//loss of precision  
        //a[4]=true; //incompatible  
        //a[4]=23.34;  
        double b[] = new double[7];  
        b[0]=10;  
        b[1]=(short)10;  
        b[2]=(long)10;  
        b[3]=(byte)10;  
        b[4]=(float)20;  
        b[5]=(int)30;  
        b[6]='a';  
        //b[7]=true;  
    }  
}
```

Variable:

Variable is a named memory location, which can be holds the data temporarily.

The data can be change from one statement to another statement.

```
int a = 10;  
a = a*20;  
a=a-100;
```

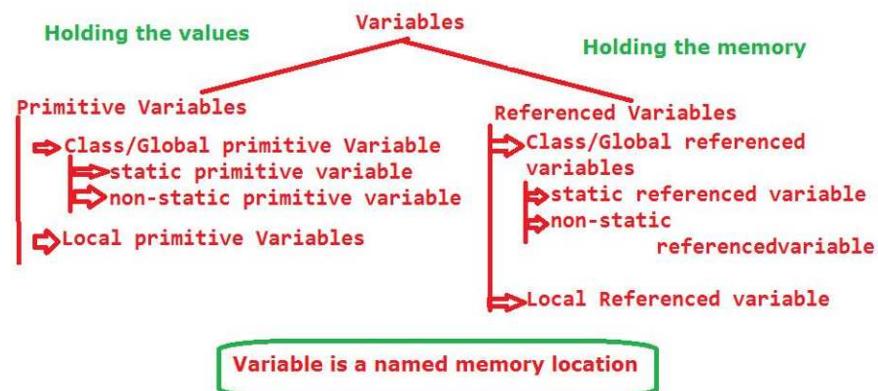
Syntax:

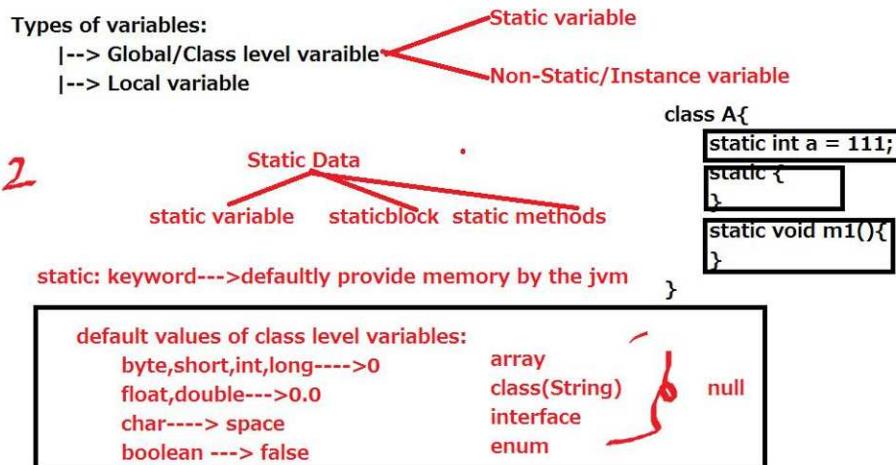
Access-Modifier modifier modifier data-type variable_name = value;

Example: public static final int a = 10;

Variable can be classified as two types

1. Primitive Variables.
2. Referenced variables.





Primitive Variable: A variable, which can be declared with the help of primitive data type is called primitive variable.

```
int x;
```

```
float y;
```

Referenced variables:

A variable which can be declared with the help of referenced datatype is called referenced variable.

```
int x[];
```

```
Student s;
```

```
Runnable r;
```

```
ArrayList al;
```

Primitive Variable:

Variable can be categorized in two types.

1. Class level variable
2. Local variable/method level variable.

Class Level Variable:

A variable, which is located at class scope those variable are called class level variable.

These are two types.

- a. static variable
- b. non-static/instance variable

a. static variables:

The variable, which have static keyword in its declaration is called static variable.

```
static int a = 111;
```

In java data is classified into two types.

Sharable data.

Non-sharable data.

Sharable data can be represents with "static" keyword.

In java we can allocate the memory in two ways

- 1. static
- 2. new

Non-sharable data comes under "new" keyword (object).

Before executing main method some static data is going to be executed.

static data means:

- 1. static variables
- 2. static blocks
- 3. static methods.

Before executing main () all static variables and static blocks will be executed, but methods are not executed automatically, if we want to execute we should call explicitly.

All the static variables having same priority, the execution priority is depend upon the way we mention in the program from top to bottom.

Variables execution means placing the data into memory.

All the static variables are memorized into two phases.

1. Loading phase.
2. Execution/initialization phase.

Loading Phase:

In this phase all the variables are memorized with default values. These default values will be placed by one special component in JVM that is "preparer".

Once Loading Phase is completed of all variables, then JVM control will goes to initialization phase.

Initialization phase:

In this phase all the default values will be replaces with original values/actual values. These original values will be given by one of the special component of JVM that is "initializer".

```
static int a = 111;
```

Initially a value is 0 in the loading phase

Later 0 will be replaced with 111 in the initialization phase.

```
java Demo  
load Demo.class
```

static loading phase	static initialization phase
<p>SV: Jvm will provide memory and filled with default value with the help of preparer.</p> <p>SB: Jvm will read heading of the block and placed into memory.</p> <p>SM: Jvm will read heading of the method and placed into memory.</p> <pre>static int a static{} static void m1(){ }</pre>	<p>SV: Jvm will replace default value with original value with the help of initializer.</p> <p>SB: Jvm wil executes static block.</p> <p>SM: Jvm will not executes the method.</p> <p>Note: completion of SLP and SIP then jvm contole goes to main method.</p> <p>a → 0 → 111</p>

```
public class StaticVariableDemo {  
  
    static int m2(){  
  
        System.out.println(b);  
  
        return 222;  
  
    }  
  
    static int a = m1();  
  
    public static void main(String[] args) {  
  
        System.out.println("main method");  
  
        System.out.println(a);  
  
        System.out.println(b);  
  
    }  
  
    static int m1(){  
  
        System.out.println(a);  
  
        return 111;  
  
    }  
  
    static int b = m2();  
  
}
```

Accessing the static variable:

We have three ways to access the static variables.

1. Class name
2. Directly (by variable name)
3. By Object or reference.

Note: If we want to call any other class static variable we have only two ways

1. Classname
2. Object or reference.

We cannot call other class static variable directly.

```
class A{  
    static int b = 222;  
}  
  
public class StaticVariableDemo {  
    static int a = 111;  
    public static void main(String[] args) {  
        System.out.println("main method");  
        int i = StaticVariableDemo.a;  
        System.out.println(i);  
        System.out.println(StaticVariableDemo.a);  
        int j = A.b;  
        System.out.println(j);  
        System.out.println(A.b);  
        System.out.println("*****");  
    }  
}
```

```

        int k = a;
        System.out.println(k);
        System.out.println(a);
        //System.out.println(b);
        System.out.println("*****");
        StaticVariableDemo sd = new StaticVariableDemo();
        System.out.println(sd.a);
        System.out.println(new StaticVariableDemo().a);
    }
}

```

Note:

static data is sharable data between Objects.

static data having only one memory location.

That memory location can be sharable by all the objects in the application.

If anyone object is updating the static data that updated data will be effected to other objects also.

<pre> communication with static data: same class static data: --> by directly --> by using class name --> by using object/reference other class static data: --> by using classname --> by using object/reference </pre>

Static block:

Block means group of statements.

A block which contains only static keyword, that block is called static block.

Syntax: static{
 }

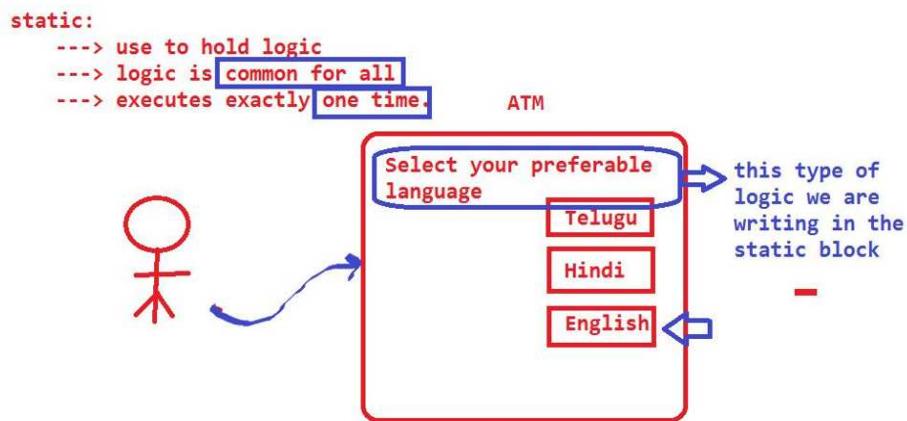
It is used for executing the common logic for all objects before main method. That type of code, we should be writes in the static block.

To initialize the static variables.

All the static blocks having same priority.

The execution of static blocks will be depends upon the way we mention in the program from top to bottom.

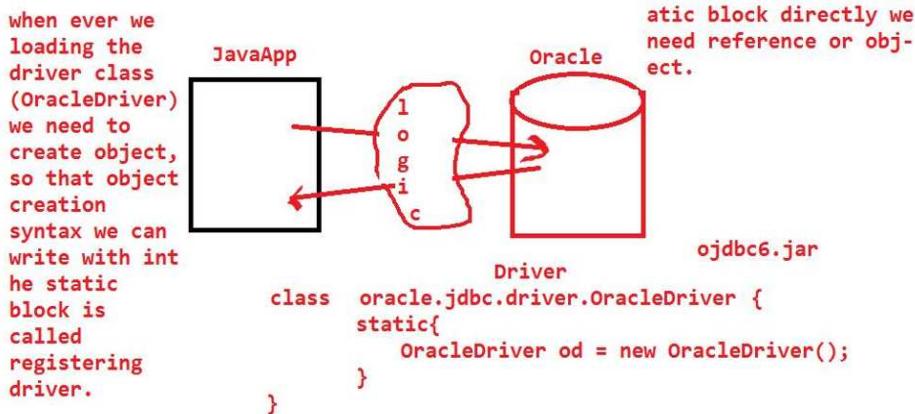
We cannot write one static block within the another static block and also another method, constructor, interface, annotation.



```

static blocks:
syntax: static{
    //logic
}
---> hold the logic.
---> common for all.
---> executes the logic exactly one time.
---> It is used to initialize the static variables
---> Registering the drivers.

```



Downloading ojdbc6/ojdbc14.jar file

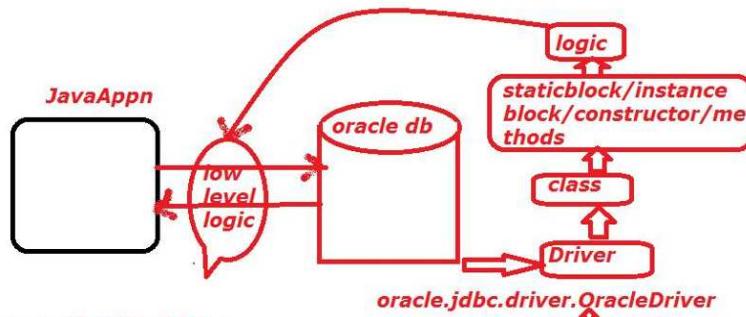
```

oracle
  jdbc
    driver
      OracleDriver

```

- > we need to load the OracleDriver class
- > we need to create Object for OracleDriver class
- > Oracle Driver class register with DriverManager
- > by using drivermanager we can get Connection between javaapp to oracle

jse-->rt.jar



```
package oracle.jdbc.driver;
public class OracleDriver{
    static{
        OracleDriver od = new OracleDeriver();
    }
}
```

```
import java.util.Scanner;
public class Atm {
    static Scanner scan = new Scanner(System.in);
    static String language;
    static byte twoDigitNumber;
    static{
        System.out.println("wel come to sbi services");
    }
    public static void main(String[] args) {
        System.out.println("main method");
    }
    static{
        System.out.println("enter your language");
        language = scan.nextLine();
        System.out.println("you are choosing language like: "
                           +language.toUpperCase());
    }
    static{
        System.out.println("enter two digit number for security");
        twoDigitNumber = scan.nextByte();
        System.out.println("you are enter number like: " +
                           twoDigitNumber);
    }
}
class A{
    static{}//valid
```

```
static{

    static{}//invalid

}

void m1(){

    static{}//invalid

}

interface I{

    static{}//invalid

}

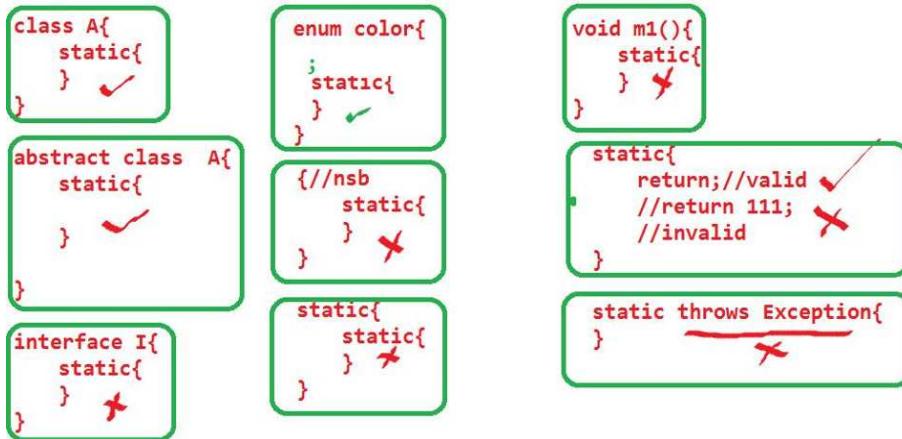
@interface AAA{

    static{}//invalid

}
```

Each and every static variables and static blocks having same priority. The priority is depends upon the way we mention in the program from top to bottom.

If both static variables and blocks are available in single class, then all static variables loading phase will be first completed. After that both static variables and static blocks will be executed (initialization phase) in the order wise. (From top to bottom).



Loading phase:

In this phase the static data is variable, then JVM provides memory, in that JVM will place default values.

If static data is block and method, then those heading will be read by the JVM and place into some memory.

Initialization phase:

In the phase, the static data is variable, then default values will be replaced with original values. If static data is block then automatically block will be executed. If static data is method then not executed, the reason is no method will be executed automatically except main.

After successfully completion of static variable, static blocks execution, then JVM will give chance to main method to execute.

static data can be sharable between all the objects. If anyone object is doing modification, that updated value effected to remaining objects also.

Note: All the static data are stored in Method Area.

Non-static data:

Non-static data comes under

non-static variable

non-static blocks
non-static methods.

Nonstatic/Instance Data

the data which not shread between all the objects is called non-static data.

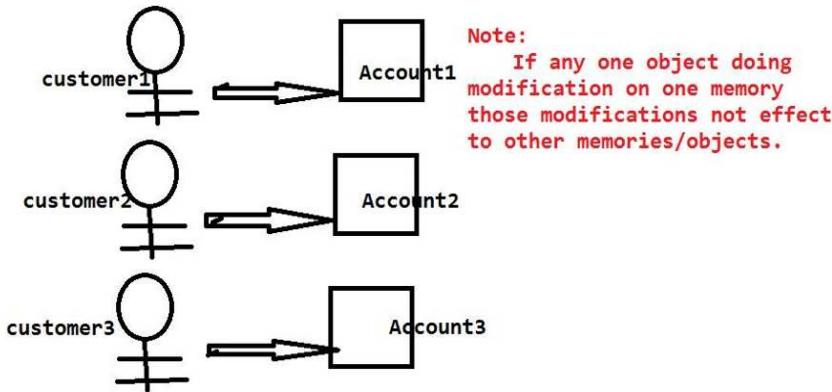
- ➡ Non-static/instance variable:
- ➡ non-static/instance blocks
- ➡ non-static/instance methods
- ➡ constructors

whenever programmer creating object for class jvm will communicating with non-static data, that jvm will provide memory for non-static data.

As many object we create, those many times non-static data will be loading in two phases



Each Every object have its own individual memory.



Non-static variable:

A variable which doesn't have any static keyword in its declaration is called non-static variable.

Ex: int a = 111;

Non-static is not sharable between any two objects.

If one object is doing modification, that updated value not effected to remaining objects.

All the non-static data and objects are stored in the heap area.

Whenever we create an object JVM will do the following things.

static data is loading or memorized only one time for entire application.

As many objects we create in our program, those many times non-static data memorized /loaded.

Loading phase:

If data is non-static variable, then JVM blindly provides memory and place the default values.

If data is non-static block, JVM will read only block heading.

If data is non-static method, JVM will read only method heading.

Initialization phase: If data is non-static variable, then jvm replace the default values with original/actual values.

If data is non-static block then block will be executed by jvm.

If data is non-static method then not executed.

All the non-static variables are having same priority, the execution of variables is depends upon, the way we mention in the program from top to bottom.

nonstatic/instance loading phase	nonstatic/instance initialization phase
<p>NSV: Jvm will give memory and filled with default value with the help of constructor.</p> <p>NSB: Jvm will read the heading and placed into memory.</p> <p>NSM: Jvm will read the heading and placed into memory.</p> <p>Constructor: Jvm will read the heading and placed into memory</p>	<p>NSV: Jvm will be replaced default values with original/actual values with the help of constructor.</p> <p>NSB: execute</p> <p>NSM: Not Execute</p> <p>Constructor: Not Execute</p> <p>Note: after succesfull completion of nonstatic loading phase and nonstatic initialization phase controle goes to constructor. <i>Once constructor is executed we can say object is sucessfully initialized.</i></p>

Communicating with Non-static/Instance data:

- 1) Same class instance data:
 - In two ways
 - |--> by directly
 - |--> by object/reference
- 2) Other class instance data:
 - only one way
 - |--> by object/reference

```
public class Nonstatic {  
    int a = m1();  
  
    int m2(){  
        System.out.println(b);  
        System.out.println("m2 method");  
        return 222;  
    }  
  
    public static void main(String[] args) {  
        System.out.println("main method");  
        Nonstatic ns = new Nonstatic();  
    }  
  
    int m1(){  
        System.out.println(a);  
        System.out.println("m1 method");  
        return 111;  
    }  
  
    int b = m2();
```

```
}
```

Note: static int a = m1();

"static int a" is comes under declaration part.

The space between equal operator and semicolon is called initialization part.

If any method calling syntax is available in initialization part, that must be carry the information from called area to calling area.

A method which is carries the information is called non-void method.

Information must be same or compatible with variable data type.

If method having return-type, that method must be have return statement with value.

Again value must be compatible with return type.

Accessing the non-static variable:

We have two ways to access the non-static data

1. By directly.
2. By object or reference.

//non-static variable we cannot call from static area directly. We should always by using reference or object.

//if we want to call any other class non-static data, we have only one way.
That is object or reference.

```
class B{  
    int b = 222;  
}  
  
public class Nonstatic {  
    int a = 111;  
    void m1(){  
        System.out.println("m1 method");  
    }  
}
```

```
        System.out.println(a);

    }

    public static void main(String[] args) {
        System.out.println("main method");
        Nonstatic ns = new Nonstatic();
        System.out.println(ns.a);
        System.out.println(new Nonstatic().a);
        ns.m1();
        //System.out.println(a);
        //System.out.println(b);
        B obj = new B();
        System.out.println(obj.b);
        System.out.println(new B().b);
    }
}
```

Non-static data is non sharable between objects. If anyone object is doing modification those modifications are not affected to other objects the reason every object have its own memory. If we are doing modification on one memory those are not affected to another memory.

```
public class C {

    int a = 111;
    int b = 222;

    public static void main(String[] args) {
        C obj1 = new C();
        System.out.println(obj1.a);
```

```
        System.out.println(obj1.b);
        System.out.println("-----");
        C obj2 = new C();
        System.out.println(obj2.a);
        System.out.println(obj2.b);
        System.out.println("-----");
        obj1.a = 888;
        System.out.println(obj1.a);
        System.out.println(obj1.b);
        System.out.println("-----");
        obj2.a = 999;
        System.out.println(obj2.a);
        System.out.println(obj2.b);
    }

}

import java.util.Scanner;
class Account{
    long accNo;
    String accHlName;
    double amount;
}

class AccountDemo{
    public static void main(String[] s){
        Scanner scan = new Scanner(System.in);
```

```
System.out.println("enter number of customers");

int noCustomers = scan.nextInt();

Account customer[] = new Account[noCustomers];

for(int i=0;i<noCustomers;i++){

    customer[i] = new Account();

    System.out.println("enter "+(i+1)+" customer details");

    System.out.println("enter accNo");

    customer[i].accNo = scan.nextLong();

    System.out.println("enter accholdername");

    customer[i].accHolName=scan.next();

    System.out.println("enter amount");

    customer[i].amount= scan.nextDouble();

}

for(int i=0;i<noCustomers;i++){

System.out.println((i+1)+" customer details");

    System.out.println(customer[i].accNo);

    System.out.println(customer[i].accHolName);

    System.out.println(customer[i].amount);

}

customer[0].amount=15000;

System.out.println(customer[0].amount);

System.out.println(customer[1].amount);
```

```
 }  
}
```

Non-static Blocks:

A block which doesn't contains static keyword in its declaration is called non-static block.

All the non-static blocks having same priority.

The execution priority is depends upon, the way we mention in the program from top to bottom.

```
public class C {  
    {  
        System.out.println("non-static block 1");  
    }  
    {  
        System.out.println("non-static block 3");  
    }  
    public static void main(String[] args){  
        System.out.println("main method");  
        C obj = new C();  
    }  
    {  
        System.out.println("non-static block 2");  
    }  
}  
  
public class C {
```

```
static {
    System.out.println("static block 1");
}

int a = m1();

{
    System.out.println("non-static block 1");
}

static int c = m3();

int m1(){
    System.out.println(a);
    System.out.println("m1 method");
    return 2222;
}

static int m3(){
    System.out.println(c);
    System.out.println("m3 method");
    return 4444;
}

{
    System.out.println("non-static block 3");
}

int b = m2();

public static void main(String[] args) {
    System.out.println("main method");
}
```

```
C obj = new C();
    System.out.println(obj.a);
    System.out.println(obj.b);
}
{
    System.out.println("non-static block 2");
}
int m2(){
    System.out.println(b);
    System.out.println("m2 method");
    return 3333;
}
}
```

Valid statement:

```
{
{
}
}

static{
{
}
}

C (){
{
```

```
    }

}

interface D{
    //{}

}

enum E{
    ;
    {

}

abstract class F{
    {

}

}

@interface I{
    //{}

}

public class C {
    {

        System.out.println("non-static block 1");
        {

    }
}
```

```
        System.out.println("non-static inner block 1 ");

    }

}

static{

    System.out.println("static block 1");

    {

        System.out.println("non-static inner block2");

    }

}

C(){

    System.out.println("constructor");

    {

        System.out.println("non-static inner block 3");

    }

}

static void m2(){

    {

        System.out.println("m2 method non-static block");

    }

}

public static void main(String[] args) {

    {

        System.out.println("main method non-static block");

    }

}
```

```

        System.out.println("mainmethod");

        C obj = new C();

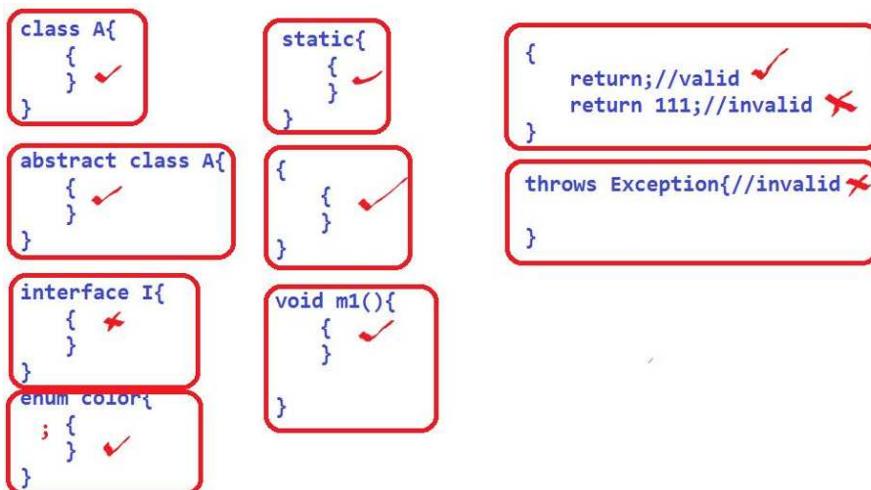
    }

}

```

Non-static block we can write with in the class, enum, abstract class, static block, static method, non-static method, constructor but we cannot write in interface and annotation.

static block we can write only within the class, enum, abstract class.



Q) Difference between static and new keyword?

static:

- 1) It is a keyword, is used to allocate memory for static data.
- 2) static data means, static variable, blocks, methods.
- 3) static data is sharable data
- 4) static data having only one memory
- 5) Only one time static data is loaded.
- 6) If any one object is doing updating on static data, that updated value will be effected to remaining objects.
- 7) Meanwhile of class loading static-data is loaded by the JVM.

8) Static data can be access with in 3 ways.

Directly, Class name, Object or reference.

9) static data we can call anywhere in the program.

10) other class static can be access only in two ways.

a.by classname

b.by object or reference

new:

1) It is keyword is used to allocate the memory for non-static data.

2) Non-static data means, non-static variables, non-static blocks, non-static methods.

We have one more name to non-static data that is instance data.

3) Non-static data is non-sharable data.

4) Non-static data having multiple memories.

5) As many object we created those many times non-static data is going to be loaded.

6) If any one object is doing updating on non-static data that updated data/value will not be effects to remaining objects.

7) Meanwhile of object creation non-static data executed.

8) Non-static can be access in two ways.

By using directly, object or reference.

9) Non-static we cannot call directly in static area.

10) Other class non-static can be access only in one way.

a. By object or reference

Nsb

sb

constructor

throw	no	no	yes
return	no	no	yes
return value	no	no	no

Local variable:

A variable, which is resides or locate within the parameter of a method, constructor and body of method, constructor, block (static, non-static) is called local variable.

Ex:

```
void m1(int x){
    int y;
}
{
    int z;
}
static{
    int l;
}
class A{
    A(int n){
        int m;
    }
}
```

x, y, z, l, m, n all are local variables.

Local variables always either default or final.

```
void m1(){  
    int a;      //valid  
    final int b; //valid  
    private int x; //invalid  
    public int y;//invalid  
    protected int z; //invalid  
    static int l; //invalid  
    transient int m; //invalid  
    volatile int n; //invalid  
}
```

Local variables may be default or final.

There are no default values for local variables.

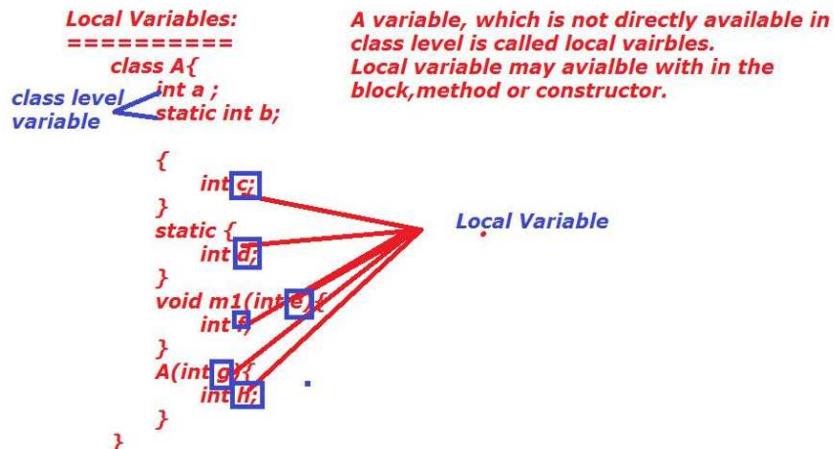
Before using local variables we should be initialized.

If we are using local variables without initialization, we will get one compile time error.

Same variable name we can declare in different scopes.

final keyword we can be applied on variables, method, classes.

Meanwhile of block execution local variables are going to be memorized and after block execution completed all the local variables are destroyed.



final variables:

A variable which has final keyword in its declaration is called final variable.

final keyword can be applied on top of class level and local variables also.

class level final variable must be initialized meanwhile of declaration.

local level final variable must be initialized before using that variable.

final variable can be used more than one time but we cannot update/change the data.

If we are trying to change we will get CE.

```

final class Student{

    String sname="suji";

}

public class FinalVariable {

    final int a = m1();

    //final int b; //class level final variable must be
    //initialize meanwhile of declaration.

    int m1(){
        
```

```
        System.out.println(a);
        System.out.println("m1 method");
        return 111;
    }

    final void m2(){
        System.out.println("m2 method");
    }

    static void m3(final int x){
        System.out.println(x);
        //x=x+1;
    }

    public static void main(String[] args) {
        FinalVariable fv = new FinalVariable();
        System.out.println(fv.a);
        //fv.a = fv.a*2;
        int c = fv.a*2;
        final int d;
        d=888;
        System.out.println(d);
        Student s = new Student();
        System.out.println(s.sname);
        s.sname="ram";
        System.out.println(s.sname);
        fv.m2();
    }
}
```

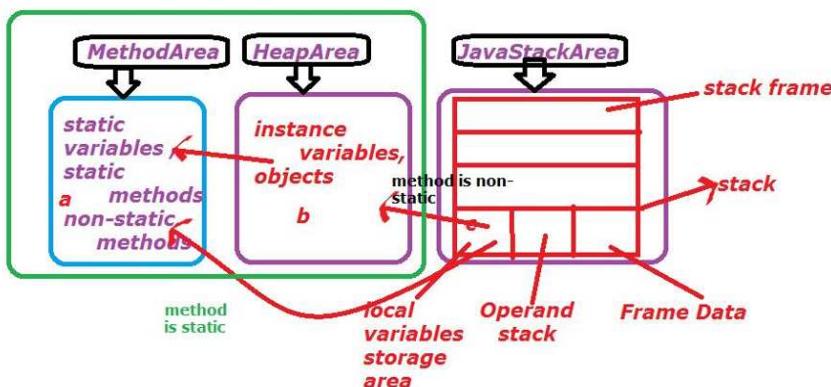
```
 }  
 }
```

Scope of variables:

Local variables: only within the block(method, constructor, static and non-static block).

Non-static variable: Based on object. once object created automatically non-static variables loaded, once object is destroyed non-static variables are destroyed.

static variable: Based on byte-code (.class).



When ever we call any variable directly first preference gives to local variable(Javastackarea).

--> If local variables is not existed then control goes to either HeapArea or MethodArea.
 --> if method is static then control goes to methodarea
 --> if method is non-static then control goes to heaparea.
 (if not available in heap area then control to method area)

=====
If we call any variable by reference or object first preference gives to HeapArea if data is not available then control goes to MethodArea.
=====

If we call any variable by using class name then control goes to MethodArea.

private, protected,public,static variables are unable to write in block level (static,non-static,constructor,method), the reason is, the main intension of

private variable intension is accessible with in the class, if we write in the block level those are not accessible with in the entire class.

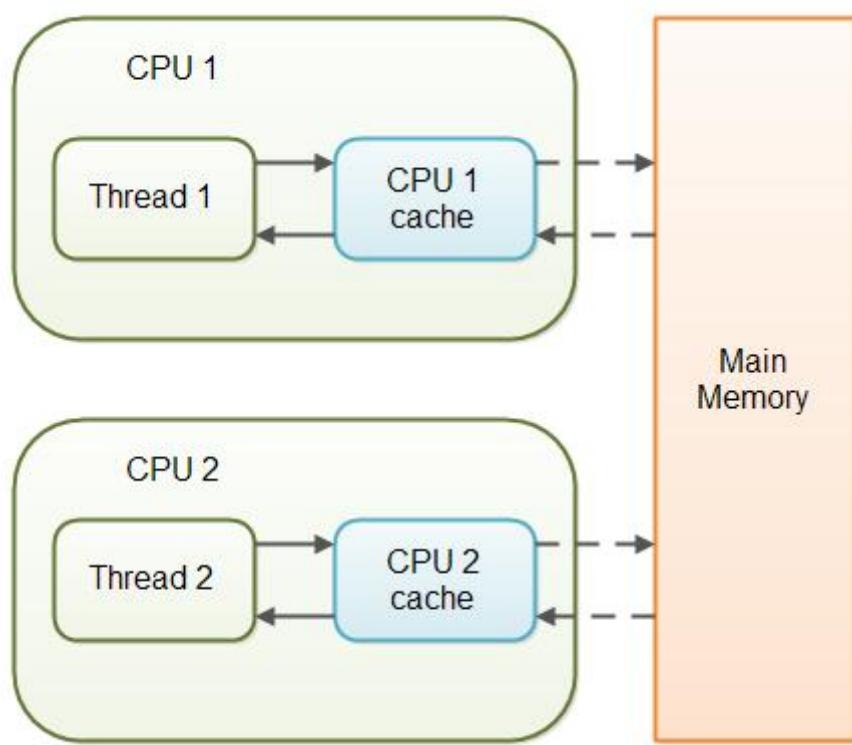
protected variable intension is accessible with in the class, and its inherited class, if we write at block level, we are unable to accessible both with in the class and its inherited classes.

public variable intension is accessible with in the any where in the project, if we declare public variable at block level, those are not available with in the project.

static variables are common for all objects(threads), if we declared at block level all threads are not accesible.

The Java volatile keyword guarantees visibility of changes to variables across threads. This may sound a bit abstract, so let me elaborate.

In a multithreaded application where the threads operate on non-volatile variables, each thread may copy variables from main memory into a CPU cache while working on them, for performance reasons. If your computer contains more than one CPU, each thread may run on a different CPU. That means, that each thread may copy the variables into the CPU cache of different CPUs. This is illustrated here:



<http://tutorials.jenkov.com/java-concurrency/volatile.html>

transient variables intention is not participated at serialization, only class level data is participated in serialization, so writing transient at block level there is no meaning.

Where shouldn't use volatile keyword?

If the variables is not shared between two threads.

When should use volatile keyword?

If the variable is shared between two threads frequently, then we should use volatile keyword.

6/2/17 * Same class, non-static / instance data:

→ by directly (from non-static area/context)
(non static or non-static method or constructor).

→ by object/reference.

Other class, non-static / instance data:

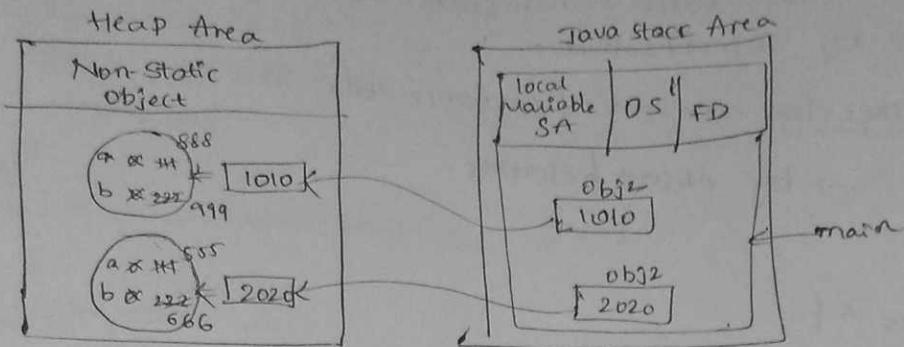
→ by object reference.

Program:

```
class A {  
    int b=222;  
}  
public class Variable {  
    int a=111;  
    {  
        S.O.P ("from nsb:" + a);  
    }  
    void m1() {  
        S.O.P ("from non-static method:" + a);  
    }  
    {  
        Variable();  
        S.O.P ("from constructor:" + a);  
    }  
    static {  
        S.O.P ("from sb:" + a);  
        S.O.P ("from sb:" + new Variable().a);  
    }  
}  
public static void main (String[] args) {  
    //S.O.P ("by directly;" + a);  
    Variable obj=new Variable();  
    S.O.P ("by reference in main:" + obj.a);  
    obj.m1();  
    A obj1=new A();  
    S.O.P ("A class B variable" + obj1.b);  
}
```

from nsb: 111
from constructor: 111
from sb: 111
from nsb: 111
from constructor: 111
by reference in main: 111
from non-static method: 111
A class B variable: 222

* If any one object is doing modification on non-static data those modifications are not effected to other objects.



Program:

```

public class Variable {
    int a=111;
    int b=222;
    public static void main (String [] args) {
        Variable obj1 = new Variable();
        Variable obj2 = new Variable();
        System.out.println ("Obj1: " + obj1.a + " " + obj1.b);
        System.out.println ("Obj2: " + obj2.a + " " + obj2.b);
        obj1.a=888;
        obj1.b=999;
        System.out.println ("=" + " = " + " = " + " = ");
        System.out.println ("Obj1: " + obj1.a + " " + obj1.b);
        System.out.println ("Obj2: " + obj2.a + " " + obj2.b);
        obj2.a=555;
        obj2.b=666;
        System.out.println ("=" + " = " + " = " + " = ");
        System.out.println ("Obj1: " + obj1.a + " " + obj1.b);
        System.out.println ("Obj2: " + obj2.a + " " + obj2.b);
    }
}

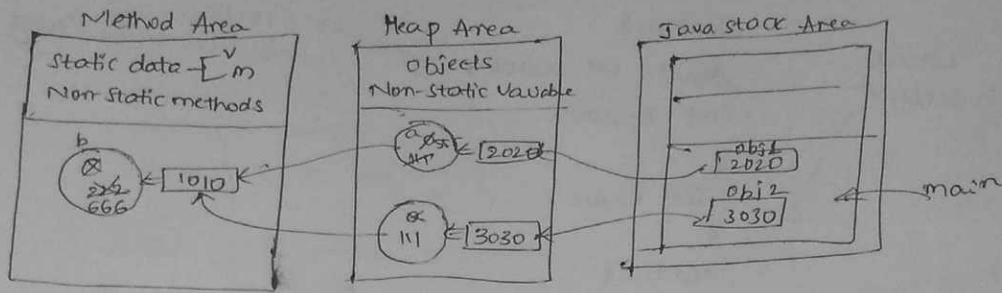
```

{OP}

111 . . . 222	888 . . . 999
111 . . . 2222	555 . . . 666
888 . . . 999	111 . . . 222
555 . . . 666	888 . . . 999

Prgm

o/p
111...222
111...222
555...666
111...666



```
public class Variable {  
    int a=111;  
    static int b=222;  
    public static void main(String[] args) {  
        Variable obj1 = new Variable();  
        Variable obj2 = new Variable();  
        System.out.println("obj1 :" + obj1.a+"..."+obj1.b);  
        System.out.println("obj2 :" + obj2.a+"..."+obj2.b);  
        obj1.a=555;  
        obj1.b=666;  
        System.out.println(" = == == ");  
        System.out.println("obj1 :" + obj1.a+"..."+obj1.b);  
        System.out.println("obj2 :" + obj2.a+"..."+obj2.b);  
    }  
}
```

Local Variable (Local primitive Variable):

A variable which is not a direct part of the class is called local variable.

```
class A {  
    static int a=100;           → CLV / GV  
    int b=200;                 (Classlevel variable / Global)  
    {  
        int c=300;  
    }  
    static {  
        int d=444;  
    }  
    A (int e){  
        int f;  
    }  
    void m1(int g){  
        int h;  
    }  
}
```

* Local variables doesn't have default values so, before using the (CLV. have) local variables we need to initialize.

Program:

```
public class Variable {  
    int a;  
    static int b;  
    public int e;  
    private int f;  
    protected int g;  
    transient int h;  
    final int i=555;  
    void m1(){  
        System.out.println("non-static m1 method");  
        int d=333;  
        System.out.println("d=" + d);  
    }  
    public static void main (String [] args) {  
        System.out.println("static main method");  
        Variable obj= new Variable();  
    }  
}
```

```

S.O.P ("MSV :" + obj1.a),
S.O.P ("SV :" + variable.b),
int c,
// S.O.P ("local variable :" + c),
obj1.m1(),
/* public int e1,
private int f1,
protected int g1;
transient int h1;
*/
final int i=666,
}
}

```

Note The scope of the local variable is within that particular block, after initialized (declaration). One block local variable we can't be used in other blocks directly.

Program

```

public class Variable {
    int m1() {
        S.O.P ("m1 method"),
        /* S.O.P ("c :" + c),
        S.O.P ("d :" + d); */
        int c=333;
        int d=444;
        S.O.P ("c :" + c);
        S.O.P ("d :" + d);
        return c;
    }
}

```

```

public static void main (String [] args) {
    S.O.P ("static main method"),
    int a=111,
    int b=222;
    Variable v=new Variable(),
    v.m1(),
}

```

```
/* s.o.p ("c"+d)  
s.o.p ("d:"+d); */
```

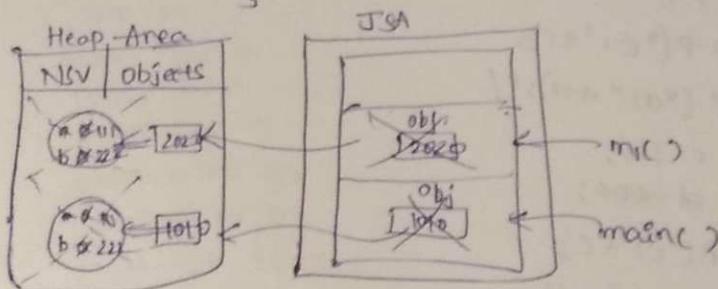
}

7/2/16

Scope of non-static Variable:

Whenever object is created, all the non-static variables getting memory or created. Once object is destroyed all the non-static variables memory destroyed that means the scope of the non-static variable is lifetime of object.

```
class A {  
    int a=111; int b=222;  
  
    void m1() {  
        A obj = new A();  
        }  
    PSVm() {  
        A obj1 = new A();  
        obj1.m1();  
        }  
}
```

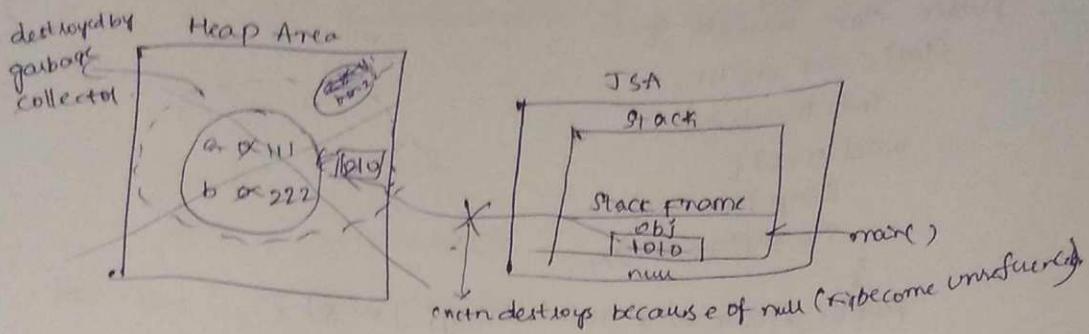


* class A {

```
    int a=111;  
    int b=222;  
    PSVm() {
```

1. A obj = new A();
2. Sop(obj.a);
3. Sop(obj.b);
4. Obj=null; // Programmer destroying un-use memory
only mem destroyed not object.

1000 . `SOP(obj.a); } If we execute again by writing these statements
SOP(obj.b) } we will get an error like,
 NULL pointer exception.`



* Class A {

```
int a=111;
int b=222;
```

```
PSVM (-) {
```

1. A obj=new A();
2. SOP(obj.a); SOP(obj.hashCode());
3. SOP(obj.b);
4. obj=null;

500 . obj=new A();
 1000 . SOP(obj.a) / cop(obj.hashCode());
 SOP(obj.b)

Note

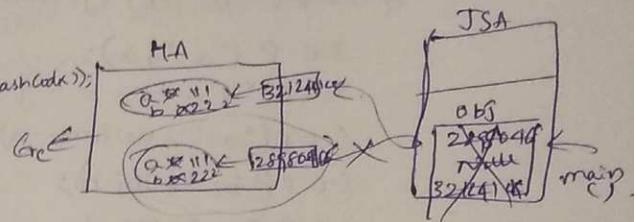
* Whenever we call any variable directly first preference gives to local variable (Java Stack Area).

→ If local variable is not existed then control goes to either heap Area or method Area.

→ if method is static then control goes to method area.

→ if method is non-static then control goes to heap area
 (if not available in heap area then control goes to method area).

* If we call any variable by reference of object first preference gives to heap area, if data is not available then control goes to method area.



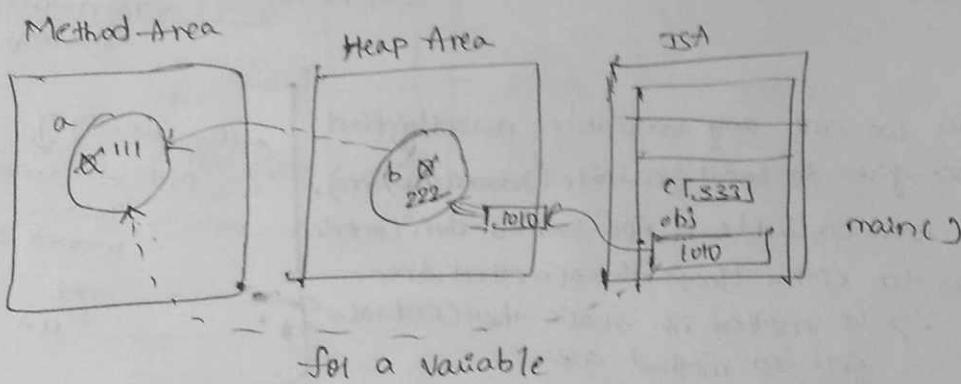
calling directly →
 preference local variable
 calling class name
 method Area

obj
 ↴ H.A.

* If we call any variable by using class name then contadeges to method area

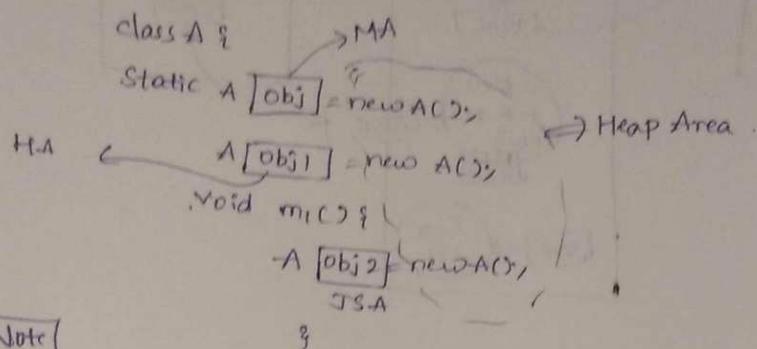
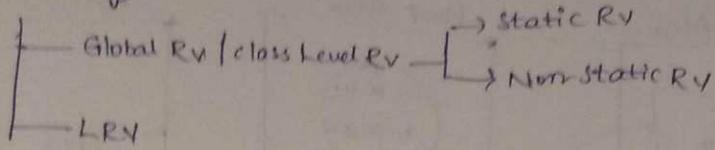
Program

```
public class variable {
    static int a=111;
    int b=222;
    void m1() {
        S.O.P ("a:" + a);
        S.O.P ("b:" + b);
    }
    public static void main(String[] args) {
        int c=333;
        variable obj=new variable();
        S.O.P ("c:" + c);
        // S.O.P ("b:" + b); // invalid // b is method is static
        S.O.P ("a:" + a);
        S.O.P ("x == == ==");
        S.O.P ("a:" + variable.a);
        S.O.P ("b:" + obj.b);
        S.O.P ("a:" + obj.a);
    }
}
```



Referenced Variables:

RV (Memory)



*Note

Reference Variable will be created in different areas but memories are always created in Heap Area.

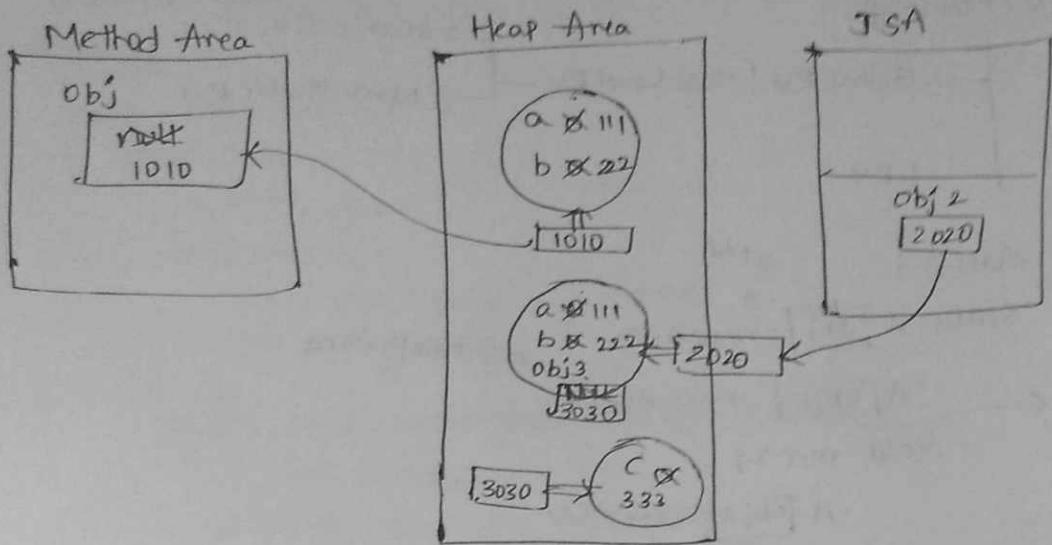
Prgrm:

```

class A {
    int c=333;
}

public class Variable {
    int a=111;
    int b=222;
    A obj3 = new A();
    //static Variable obj=newVariable();
    Variable obj1 = new Variable();
    public static void main(String[] args) {
        System.out.print("*****");
        Variable obj2 = new Variable();
    }
}
  
```

Internal-flow:



Referenced Variables:

Referenced variables are categorized into two types.

1. Class level referenced variable
2. Local referenced variable.

Class level referenced variable:

A referenced variable available at class level is called CLRV.

It has been categorized into two types

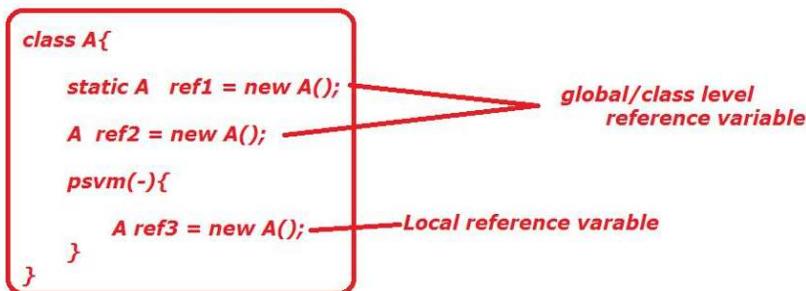
- a. static referenced variable
- b. non-static/instance referenced variable.

```
class Student{  
    Student s = new Student();  
    static Student s1 = new Student();  
}
```

Referenced Variable:
(to hold memory---->have multiple values)

-->class/global referenced variables
-->static referenced variables
-->instance referenced variables
-->local referenced variables

```
ClassName ref = new ClassName();  
Student s = new Student();  
          ↓  
reference variable
```



Local referenced variable:

A variable which is available at block or method level is called LRV.

Ex:

```
class Student(){  
    p s v main(-){  
        Student s = new Student ();  
    }  
}  
  
package oops;  
  
public class Variable {
```

```

int a = 111;
static Variable obj2 = new Variable();
//Variable obj = new Variable();

public static void main(String[] args) {
    System.out.println("*****");
    Variable obj1 = new Variable();
}
}

/*Note: the combination of bellow statements at class level
illegal
static Variable obj2 = new Variable();
Variable obj = new Variable();
*/
/* we can not write following statements, two times at class level
and method level illegal
Variable obj = new new Variable();
*/

```

Arrays can hold more than one value with same type, but it is unable hold different type of data , to overcome this problem then we can go for another referenced datatype is called "class".

Class is an imaginary thing, which is not existed in the real world.

Class is a model.

Class is a model for creating objects. Means the properties and actions of the objects are written in the class.

Properties are represented by variables.

Actions of the objects are represented by methods.

So a class contains variables and methods.

The same variable and methods are also available in the objects because they are created from the class.

These variables are also called as instances.

Q) How can we provide functionality to a class?

A) By creating an object.

Object:

Whatever the functionalities the class having, we should get those functionalities by creating object only.

Creating an object is nothing but, allocating memory, necessary to store the actual data of the variable.

By creating the object only, we can give the memory to name, age variables

Class Def:

It is a java programming element. It can be hold both static and non-static variables, static and non-static method, static and non-static block, both static and non-static inner class, interface, enum.

Object Creation:

In java we have 6 ways to create an object.

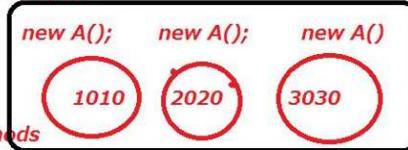
1. new keyword
2. clone()
3. newInstance()
4. Factory methods.
 - a. static factory methods
 - b. non-static factory methods
5. Deserialization.
6. Literals.

Object Creation:

Allocating the memory for non-sharable/non-static-instance data.

There are 5 ways to create object in java

- 1) new
- 2) newInstance()
- 3) clone()
- 4) FactoryMethods
 - 4.a) Static FactoryMethods
 - 4.b) Non-Static Factory Methods
- 5) Deserialization.



As many new keywords we use, those many new memories are allocating by the jvm.

For each every execution these memories are going to be and change.

1. new Keyword:

In java we can create two types of objects.

- a. referenced object.
- b. un-referenced object.

Referenced Object Creation or Used Memory:

Syntax: Class_name reference_name = new class_name();

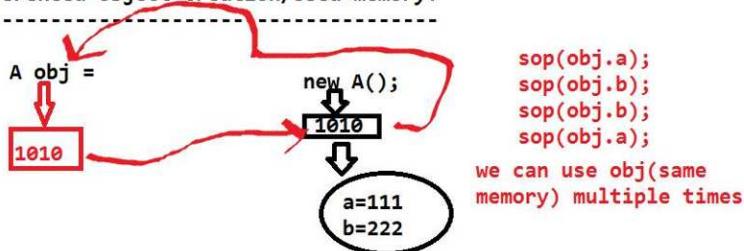
Whenever JVM see the new keyword blindly provides new memory.

With the help of referenced object creation we can reuse the memory more than one time.

with the of unreferenced object creation we can communicate with the memory only one time.

If we want to communicate with the memory multiple times then we can go for referenced object creation.

Referenced Object creation/Used memory:



Unreferenced object creation:

```
new class_name();
```

With help of unreferenced object we can use the memory only one time.

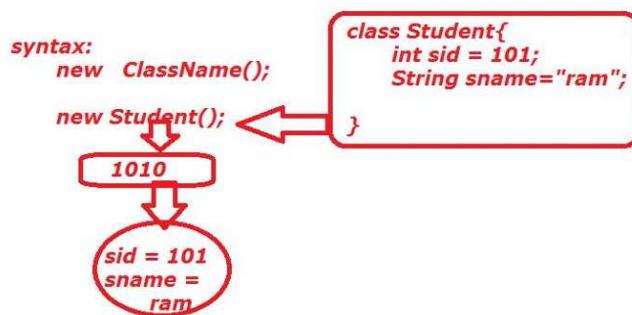
With the help of object we can communicate with both static and non-static data also.

```
class Student {  
    int a = 111;  
    int b = 222;  
    psv main(){  
        Student s1 = new Student();  
        new Student();  
    }  
}
```

In java we can create two types of objects.

1) Referenced object creation/Used memory

2) Unreferenced object creation/Unused memory



Difference between object and reference:

Object means allocating memory.

Reference means holding the memory, which is generated by object.

But both object and reference will points to same memory.

With the help of reference, we can use the same memory more than one time.

With the help of object, we can use the memory only one time.

```
public class SRV {  
    int x = 100;  
    int y = 200;  
  
    public static void main(String[] args) {  
        SRV obj = new SRV();  
        System.out.println(obj.x);  
  
        System.out.println(obj.y);  
        new SRV();  
        int a = new SRV().x;  
        int b = new SRV().y;  
        System.out.println(new SRV().x);  
        System.out.println(new SRV().y);  
    }  
}  
  
public class SRV {  
    SRV obj = new SRV();  
  
    public static void main(String[] args) {  
        SRV obj1 = new SRV();  
    }  
/*static SRV obj = new SRV();
```

```
public static void main(String[] args) {  
    SRV obj1 = new SRV();  
    System.out.println(obj1);  
    System.out.println(obj);  
}*/  
}  
}
```

```
public class SRV {  
    int a = 111;  
    static int b = 222;  
    public static void main(String[] args) {  
        int a = 333;  
        int b = 444;  
        SRV obj = new SRV();  
        System.out.println(a);  
        System.out.println(obj.a);  
        System.out.println(obj.b);  
        System.out.println(b);  
    }  
}
```

JVM always gives priority to local data.

If we are calling variable directly first compiler will within the local area (block), if not then it will go to class level. (Heap or Method area).

s.o.p(obj.a) then compiler directly check in object are(heap area),f available that a value bind to obj, then JVM will print a value in runtime.

If not compiler will check in the method area or static are

If available compiler will bind the data to obj, then JVM will print a value in runtime.

If not available then compile time error.

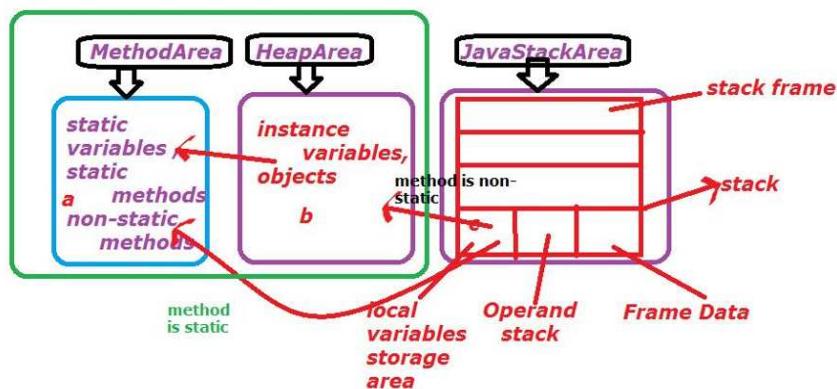
```
public class NSD {  
    static int b = 333;  
    int a = 111;  
    //static int a = 222;  
    //same we cannot be define more than one time  
    //in the same scope  
    public static void main(String[] args) {  
        int a = 222;  
        System.out.println(a);(1)  
        new NSD();//no use  
        System.out.println(new NSD().a);(2)  
        NSD ns = new NSD();  
        System.out.println(ns.a);(3)  
        System.out.println(ns.b);(4)  
        System.out.println(NSD.b);(5)  
        System.out.println(b);(6)  
    }  
}
```

Note:

In step 1) JVM will give priority to local data first

If data is not available in block level (Java Stack Area) then control goes to method area.

- 2) JVM control directly goes to heap area if not available then goes to method area
- 3) Same as.
- 4) Same as 2.
- 5) Same control directly goes to method area
- 6) JVM control goes to Java Stack Area, if not available then control to goes to method area



Scopes of variables:

We have two scopes.

1. Class scope
2. Block scope

Class scope variables we can access within the entire class level

Block scope variables we can access within the same scope.

```
class Student1{
```

```
    int sid = 111;
```

```
        String sname = "suji";  
    }  
  
public class SRV {  
  
    public static void main(String[] args) {  
  
        Student1 s1 = new Student1();  
  
        Student1 s2 = new Student1();  
  
        Student1 s3 = new Student1();  
  
        System.out.println("s1: "+s1.sid+"...."+s1.sname);  
  
        System.out.println("s2: "+s2.sid+"...."+s2.sname);  
  
        System.out.println("s3: "+s3.sid+"...."+s3.sname);  
  
    }  
  
}
```

Output:

s1: 111....suji

s2: 111....suji

s3: 111....suji

In above program we are getting same id, name for all the students, it is not possible in real time, to avoid this drawback we should provide different values for different students. If we want to achieve this requirement then we can go for constructors.

Constructors:

Constructors are special blocks in class level, to initialize the non-static variables meanwhile of object creation.

Constructors are used to creating object.

Constructors are class level blocks.

Constructor name is same as class name.

After successfully executing of all non-static variables, blocks then constructors will be executed.

Once constructor is successfully executed, then we can say this, our object is successfully initialized.

Constructors are plays major role in object creation.

In constructor block we can write throws keyword and return statement without value. But we cannot write return statement with value.

Without constructor we cannot create object for a class "with new keyword".

Constructors are three types.

- a. Default constructor.
- b. Zero argument/non-parameterize constructor.
- c. Parameterized/ augmented constructor.

Default constructor:

If we are not specify any constructor in our class by default compiler will provide one constructor with zero parameters is called default constructor.

In that default constructor compiler will provides one non-static variable that is "super ()".

Ex: class A{

}

javac A.java

javap A

```
class A extends java.lang.Object{
```

```
    A(){
```

```
    super();  
}  
}
```

javap command is used to check the .class files information.

javap command always needs fully qualified name(package name + class name /interface name/ enum /abstract class/ annotation).

If our class is public compiler generated constructor also public.

If our class is default compiler generated constructor also default.

We cannot write protected, private, static, native, volatile, transient, synchronized modifiers in front of outer class name.

We can write only default, public, final, abstract, strictfp in front of the outer class names.

Zero-argument constructor:

The constructor, which is created by the programmer without any argument is called zero argument constructor.

```
public class A{  
    A(){//zero argument  
}  
}
```

Zero argument constructor can allows, all the access modifier (private, default, protected, public)

Zero argument constructor is same as default constructor in two areas (default, public), in remaining two areas both are different.

Note:

If the class is default, then compiler provide constructor (default constructor) and zero argument constructor with default access modifier

then only both default and zero argument constructor are same, otherwise different.

Programmer can write private,default,protected public constructors.

Compiler will provide default and public constructors.

	programmer	compiler
default constructor	Yes	Yes
public constructor	Yes	Yes
private constructor	Yes	No
protected constructor	Yes	No

Note: zero argument and default constructor are same at only in two places those are default and public in remaining cases both are different

Parameterized constructor:

The constructor, which is created by the programmer with argument/parameters is called parameterized constructor.

The parameter count must be at least one.

Ex:

```
class Student{  
    Student(){}//zero argument  
}  
  
Student(int x){//parameterized  
}  
}
```

Within the one class we can write multiple constructors.

Q) Difference between method and constructor?

Method name and constructor name can be same as class name, but method always having return type, whereas constructor doesn't.

Method having identity where as constructor doesn't have.

Constructors are automatically executed meanwhile of object creation.

Whereas methods never executed automatically meanwhile of class loading as well as object creation. If we want to execute method we need to call.

```
class Student1{  
    int sid = m1();  
  
    int m1(){  
        System.out.println(sid);  
        System.out.println("m1 method");  
        return 111;  
    }  
  
    Student1(){  
        System.out.println("zero argument constructor");  
    }  
    {  
        System.out.println("non-static block");  
    }  
  
    Student1(int x){  
        System.out.println("parameterized constructor");  
        System.out.println(x);  
    }  
}  
  
public class SRV {  
    public static void main(String[] args) {  
        Student1 s1 = new Student1();
```

```
        Student1 s2 = new Student1(200);  
    }  
}
```

For single object creation only one respectable constructor is executed.

If we want create an object we need respective constructor otherwise we will not create object with the help of new keyword.

As many object we created those many times non-static variables, blocks, constructors (respective) will be executed.

Object creation means communicating with the constructor only for initializing the non-static data/variables and forwarding the control from sub class to super class with the help of super keyword.

At time we can call only one constructor.

Constructor overloading:

Writing the same constructor more than one time with in the same class with different parameters is called constructor overloading.

```
class A{  
    A(){  
    }  
    A(int x){  
    }  
    A(float y){  
    }  
    A(int x, float y){  
    }  
    A(float x, int y){  
    }  
}
```

Note: Don't consider variable names.

Note: one class constructor can't be placed in another class.

```
class A{  
    B(){//invalid
```

```
    }  
}  
  
class B{  
}
```

Executing the non-static variable and blocks is called object creation.

Executing constructor is called object initialization

Why should we go for Constructor Overloading?

```
A)  
class A{  
    int a[];  
    A(){  
        new int[10];  
    }  
    A(int x){  
        new int[x];  
    }  
  
    public static void main(String[] r){  
        A obj = new A();  
        A obj1 = new A(20);  
    }  
}
```

```
import java.util.Scanner;  
  
class Account{  
    double totalBalance;  
  
    Account(){  
    }  
  
    Account(double minimumBalance){  
        totalBalance = minimumBalance;  
    }  
  
    public static void main(String[] s){  
        System.out.println("enter type of person");  
        Scanner scan = new Scanner(System.in);  
    }  
}
```

```

String personType = scan.next();
if(personType.equalsIgnoreCase("Student")){
    Account obj1 = new Account();
    System.out.println("balance: "+obj1.totalBalance);
}
else
    if(personType.equalsIgnoreCase("Employee")){
        Account obj2 = new Account(5000);
        System.out.println("balance: "+obj2.totalBalance);
    }
    else {
        Account obj3 = new Account(500);
        System.out.println("balance: "+obj3.totalBalance);
    }
}

}

```

Constructor chaining:

Communicating with one constructor to another constructor is called constructor chaining.

If we want develop constructor chaining in java, we need any one of these keywords

1. this.

2. super.

constructor chaining:

Making a communication between one constructor to another.

communication between

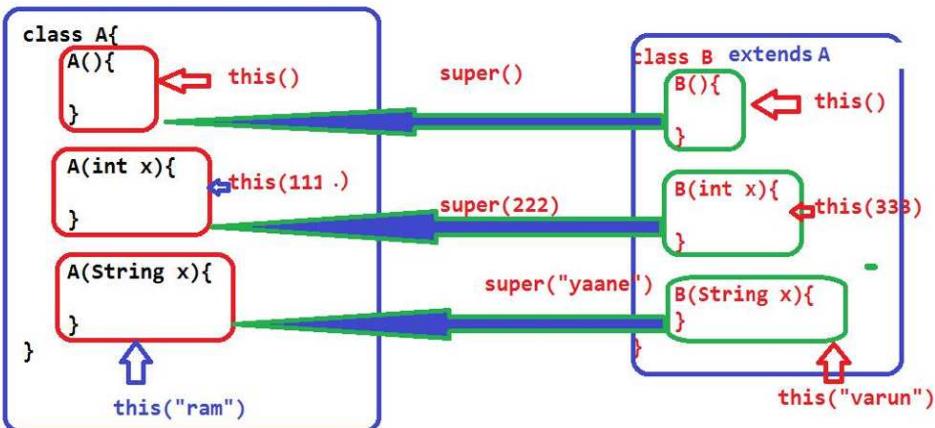
subclass constructor--> super class constructor(super())

one constructor to another--->`this()`

we can represents constructors

A()	<code>this()</code>	B() ---> <code>super()</code>
A(int x)	<code>this(111)</code>	B(int x) ---> <code>super(111);</code>
A(String x)	<code>this("ram");</code>	B(String x) ---> <code>super("anjali")</code>
A(int x, float y())	<code>this(100,34.00f)</code>	B(int x, float y)---> <code>super(100,20f)</code>
A(float y, int x)	<code>this(34f,2000)</code>	B(float y, int x)---> <code>super(1f,2)</code>

ways
 How many can we communicate with constructor?
 --> new keyword
 --> this()
 --> super()



```

class Student{
  int sid;
  String sname;
  Student(){
    this(111);
    System.out.println("zero argument constructor");
  }
  Student(int x){
    this("ram");
    System.out.println("single int parameterized constructor");
  }
}
  
```

```

        System.out.println(x);
    }

    Student(String s){
        System.out.println("single String parameterized constructor");
        System.out.println(s);
    }

}

public class ConstructorChaining {
    public static void main(String[] args) {
        Student s = new Student();
    }
}

class B{
    B(int x){
        System.out.println("super class int argument");
    }
}

class C extends B{
}

```

In the above code we will get compile time error.

The reason is in the C class compiler will provide default constructor and super () (zero argument).

Like as bellow

```
class C extends java.lang.Object{
```

```
C(){  
    super();  
}  
}
```

But in the super class we don't have any zero argument constructors.

Note: If super class not having any zero argument constructor, in the sub class programmer must be write constructor and with appropriate super () with argument.

```
package reference;  
  
class B{  
    B(){  
        System.out.println("super class int argument");  
    }  
}  
  
class C extends B{  
    private C(){  
    }  
    public static C getC(){  
        return new C();  
    }  
    int a = 111;  
}  
  
public class ConstructorChaining {  
    public static void main(String[] args) {  
        //C obj = new C();
```

```

        C obj1 = C.getC();
        System.out.println(obj1.a);
    }
}

```

If any class having private constructor, that class must be have static factory method to communicate its(class) non-static data.

Singleton Design pattern:

A design patterns are well-proved solution for solving the specific problem/task.

SDP represents allocating memory for a particular class only one time. The same memory can be use multiple times. With the SDP we can save the memory.

If we want to develop SDP class we need one private constructor and public static reference variable and one static factory method.

*Singleton design pattern:
providing only one memory for one class is called singleton.*

*-->private constructor
-->static factory methods
-->static referenced object creation*

```

class Servlet{
    public static Servlet obj = new Servlet();
    private Servlet(){}
    public static Servlet getServlet(){

```

```

        return obj;
    }

}

public class SingletonDesignPattern {
    public static void main(String[] args) {
        //System.out.println(new Servlet());

        Servlet obj1 = Servlet.getServlet();
        Servlet obj2 = Servlet.getServlet();
        System.out.println(obj1.hashCode());
        System.out.println(obj2.hashCode());
    }
}

```

Additional Program on constructor chaining:

```

package variable;

class A{
    A(){
        System.out.println("super class zero argument constructor");
    }
    A(int x){
        System.out.println("super class int argument constructor");
    }
    A(String x){
        System.out.println("super class String argument constructor");
    }
}
class B extends A{
    B(){
        //super();
        System.out.println("sub class zero argument constructor");
    }
    B(int x){
        System.out.println("sub class int argument constructor");
    }
    B(String x){
        System.out.println("sub class String argument constructor");
    }
}

```

```

        }
    }
public class ConstructorChaining {
    public static void main(String[] args) {
        new B();
    }
}
o/p:
super class zero argument constructor
sub class zero argument constructor

package variable;

class A{
    A(){
        System.out.println("super class zero argument constructor");
    }
    A(int x){
        System.out.println("super class int argument constructor");
    }
    A(String x){
        System.out.println("super class String argument constructor");
    }
}
class B extends A{
    B(){
        this(123);
        System.out.println("sub class zero argument constructor");
    }
    B(int x){
        //super();
        System.out.println("sub class int argument constructor");
    }
    B(String x){
        System.out.println("sub class String argument constructor");
    }
}
public class ConstructorChaining {
    public static void main(String[] args) {
        new B();
    }
}

package variable;

class A{
    A(){
        //super();
        System.out.println("super class zero argument constructor");
    }
    A(int x){
        System.out.println("super class int argument constructor");
    }
    A(String x){
        System.out.println("super class String argument constructor");
    }
}

```

```

        }
    }
class B extends A{
    B(){
        this(123);
        System.out.println("sub class zero argument constructor");
    }
    B(int x){
        this("ram");
        System.out.println("sub class int argument constructor");
    }
    B(String x){
        //super();
        System.out.println("sub class String argument constructor");
    }
}
public class ConstructorChaining {
    public static void main(String[] args) {
        new B();
    }
}

package variable;

class A{
    A(){
        this(234);
        System.out.println("super class zero argument constructor");
    }
    A(int x){
        this("varun");
        System.out.println("super class int argument constructor");
    }
    A(String x){
        System.out.println("super class String argument constructor");
    }
}
class B extends A{
    B(){
        this(123);
        System.out.println("sub class zero argument constructor");
    }
    B(int x){
        this("ram");
        System.out.println("sub class int argument constructor");
    }
    B(String x){
        //super();
        System.out.println("sub class String argument constructor");
    }
}
public class ConstructorChaining {
    public static void main(String[] args) {
        new B();
    }
}
```

```

}

package variable;

class A{
    A(){
        this(234);
        System.out.println("super class zero argument constructor");
    }
    A(int x){
        this("varun");
        System.out.println("super class int argument constructor");
    }
    A(String x){
        super();
        System.out.println("super class String argument constructor");
    }
}
class B extends A{
    B(){
        this(123);
        System.out.println("sub class zero argument constructor");
    }
    B(int x){
        this("ram");
        System.out.println("sub class int argument constructor");
    }
    B(String x){
        //super();
        System.out.println("sub class String argument constructor");
    }
}
public class ConstructorChaining {
    public static void main(String[] args) {
        new B();
    }
}

package variable;

class A{
    A(){
        this(234);
        System.out.println("super class zero argument constructor");
    }
    A(int x){
        this("varun");
        System.out.println("super class int argument constructor");
    }
    A(String x){
        super();
        System.out.println("super class String argument constructor");
    }
}
class B extends A{
    B(){

```

```

        this(123);
        //super();

        System.out.println("sub class zero argument constructor");
        //this(123);
        //super();
    }
    B(int x){
        this("ram");
        System.out.println("sub class int argument constructor");
    }
    B(String x){
        System.out.println("sub class String argument constructor");
    }
}
public class ConstructorChaining {
    public static void main(String[] args) {
        new B();
    }
}

```

Copy Constructor:

Placing one object data into another object is called copy constructor.

The constructor which have same class reference variable as parameter is called copy constructor.

```

package inheritance;

class Student{

    int sid ;

    String sname;

    int sage;

    Student(int sid,String sname,int sage){

        this.sid = sid;

        this.sname = sname;
    }
}

```

```

        this.sage = sage;
    }

    Student(Student obj){//copy constructor

        this.sid = obj.sid;
        this.sname = obj.sname;
        this.sage = obj.sage;
    }

}

public class CopyConstructor {

    public static void main(String[] args) {

        Student s = new Student(101,"mounika",25);

        Student s1 = new Student(s);

        System.out.println("s: "+s.sid+"..." +s.sname+".." +s.sage);

        System.out.println("s1:"+s1.sid+"..." +s1.sname+".." +s1.sage);
    }

}

```

"this" keyword:

"this" is a non-static final reference variable, which can be created by compiler and memory filled by the JVM, meanwhile of object creation, to hold current object.

Current Object means, the object which is used by the programmer currently/at present.

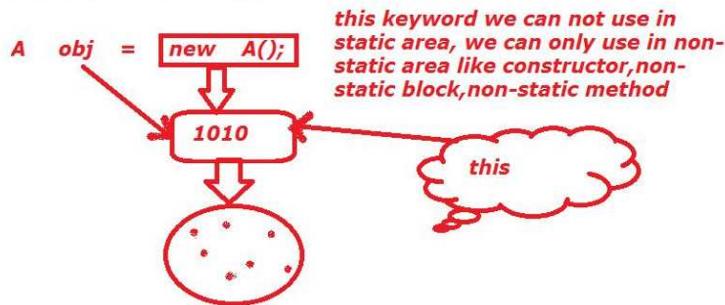
- 1) this keyword is a non-static final reference variable.
- 2) this keyword can't be used from static area(static method, static block). It can be accessed only within the non-static area(non-static method, non-

static block, constructor). We can't access this keyword even by using object also.

this keyword:

this is a non-static final reference variable, created by the jvm meanwhile of object creation.

this keyword is holding current object.



```
public class ThisDemo {  
    int a = 111;  
    public static void main(String[] args) {  
        ThisDemo td = new ThisDemo();  
        System.out.println(td.a);  
        //System.out.println(this.a);  
    }  
}
```

3) We cannot change the value of this keyword.

```
public class ThisDemo {  
    int a = 111;  
    void m1(){  
        System.out.println(this);  
        //System.out.println(this+100);
```

```

}

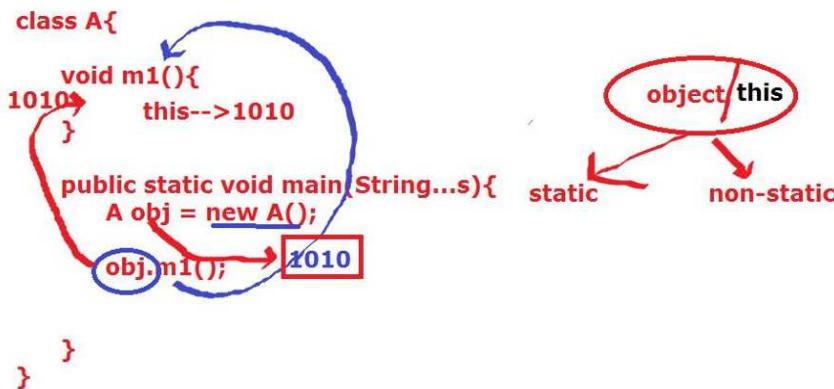
public static void main(String[] args)      {
    ThisDemo td = new ThisDemo();
    System.out.println(td.a);
    //System.out.println(this.a);
    td.m1();
}

}

```

- 4) Whenever we calling m1 () on top of "td" reference (current object), the m1 () is going to be executed, in the meanwhile td reference value is also going to m1 ().

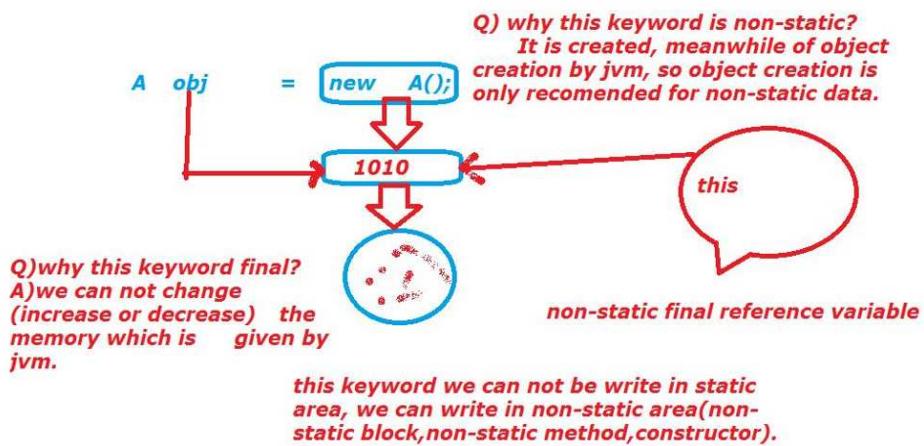
If we want to hold td reference value in m1() we need "this" keyword.



- 5) this keyword is also communicating with both static and non-static data.
- 6) this keyword is used to make a communication between constructor, with in the same class.
- 7) this keyword must be first statement in the constructor.
- 8) We should leave one constructor for "super" keyword.

That means we can't use this keyword in all constructor.

9) We can't call same constructor, within its body.



```
class Student{  
    Student(){  
        this(111);  
        System.out.println("zero arguement constructor");  
        //this();  
    }  
    Student(int x){  
        //this();  
        System.out.println("parmeterized constructor");  
    }  
}  
public class ThisDemo {  
    int a = 111;  
    static int b = 222;  
    void m1(){
```

```
        System.out.println(this.a);
        System.out.println(this.b);
    }

    public static void main(String[] args) {
        ThisDemo td = new ThisDemo();
        td.m1();
        Student s = new Student();
    }

}

public class ThisDemo {
    int a = 111;
    static int b = 222;
    static {
        System.out.println("static block");
        //System.out.println(this);
    }
    ThisDemo(){
        System.out.println("constructor: "+this);
    }
    {
        System.out.println("non-static block");
        System.out.println("nsb:***** "+this);
    }
    void m1(){

```

```

        System.out.println("m1: "+this);
        //System.out.println(this++);
        System.out.println("m1 method");
        System.out.println(this.a);
        System.out.println(this. b);
    }

    public static void main(String[] args) {
        System.out.println("main method");
        //System.out.println(this);
        ThisDemo td = new ThisDemo();
        System.out.println("td: "+td);
        td.m1();
        System.out.println("*****");
        ThisDemo td1 = new ThisDemo();
        System.out.println("td1:"+td1);
        td.m1();
    }
}

```

10) this keyword is used forwarding the current object to another method as an argument.

```

class Student{
    void m1(){
        System.out.println("m1 method");
        System.out.println("this: "+this);
    }
}

```

```
m2(this); //compiler conver into this.m2(this);

System.out.println("-----");
Student s1 = new Student();
System.out.println("s1: "+s1);
m2(s1); //compiler conver into this.m2(s1);

System.out.println("-----");
m2(new Student()); //compiler convert into// this.m2(new Student());
System.out.println("-----");
m2(this);

}

void m2(Student s){

    System.out.println("m2 method s: "+s);

}

}

public class ThisDemo {

    public static void main(String[] args) {

        Student s = new Student();

        System.out.println("s: "+s);
        s.m1();

    }

}
```

11) this is used to differentiate class level variable to local variable.

```
class Student{

    int sid = 111;
```

```
String sname = "ram";  
/*Student(int id, String name){  
    sid = id;          //this.sid = id;  
    sname = name;//this.sname = name;  
}*/  
/*Student(int sid, String sname){  
    sid = sid;  
    sname = sname;  
}*/  
Student(int sid, String sname){  
    this.sid = sid;  
    this.sname = sname;  
}  
}  
  
public class ThisDemo {  
    public static void main(String[] args) {  
        Student s = new Student(222,"suji");  
        System.out.println(s.sid);  
        System.out.println(s.sname);  
    }  
}  
  
package variable;  
  
public class L {  
    L(){  
}
```

```

        System.out.println("constructor:this : "+this.hashCode());
    }
{
    System.out.println("non-static block:this: "+this.hashCode());
}

public static void main(String[] args) {
    System.out.println("main method");
    L obj = new L();
    System.out.println("main:obj:           : "+obj.hashCode());
    System.out.println("=====");
    L obj1 = new L();
    System.out.println("main:obj1:           : "+obj1.hashCode());
}
}

```

If any non-static context(non-static block, constructor, non-static method) executing with the help of any object, that object only the current object in that particular scenario, so "this" keyword always pointing to that object only.

super keyword:

super keywords is an non-static final reference variable, to make communication between subclass constructor to super class constructor.

Whenever using extends keyword whatever the data which is available in super class that data will comes to subclass. If we want extract the data first we need to forward our control from subclass to super class.

For this purpose we need one predefine keyword that is "super".

If we are not writing any super (), this () in constructor then compiler will place super (), in our constructor.

With in the single constructor we can write either this () or super (), we cannot write both. The reason is both super () and this () always needs first statement in the constructor.

All static and non-static variable, static and non-static block and constructor are loaded from super class to sub class.

```
class A extends java.lang.Object{  
    int a = m1();  
    static int b = m2();  
    int m1(){  
        System.out.println(a);  
        System.out.println("A class non-static m1 method");  
        return 111;  
    }  
    A(){  
        super();  
        System.out.println("A class constructor");  
    }  
    static{  
        System.out.println("A class static block");  
    }  
    {  
        System.out.println("A class non-static block");  
    }  
    static int m2(){  
        System.out.println(b);  
    }  
}
```

```
        System.out.println("A class static m2 method");

        return 222;

    }

}

class B extends A{

    int c = m3();

    static int d = m4();

    int m3(){

        System.out.println(c);

        System.out.println("B class non-static m3 method");

        return 333;

    }

    B(){

        super();

        System.out.println("B class constructor");

    }

    B(int x){

        super();

        //this();




        System.out.println("B class argument constructor");

    }

    static{

        System.out.println("B class static block");

    }

}
```

```
}

{

    System.out.println("B class non-static block");

}

static int m4(){

    System.out.println(d);

    System.out.println("B class static m4 method");

    return 444;

}

}

public class SuperDemo {

    public static void main(String[] args) {

        B obj = new B();

    }

}
```

"super" is used to differentiate the subclass variable to super class variable.

"super" is used to differentiate the subclass method to super class method.

"super" keyword can't used in static area.

With the help of super keyword we can call both static and non-static data.

super keyword always represent super class memory.

```
package constructor;

class B{
```

```
int a = 111;
static int b = 222;
void m1(){
    System.out.println("B class non-static m1()");
}
static void m2(){
    System.out.println("B class static m2()");
}
}

public class SuperDemo extends B{
    int a = 333;
    static int b = 444;
    void m1(){
        int a = 555;
        System.out.println("SuperDemo class non-static m1()");
        System.out.println(a);
        System.out.println(this.a);
        System.out.println(super.a);
        System.out.println(b);
        System.out.println(this.b);
        System.out.println(super.b);
        m2(); //this.m2();
        this.m2();
        super.m2();
    }
}
```

```

super.m1();

}

static void m2(){

    System.out.println("SuperDemo class static m2()");

}

public static void main(String[] args) {

    //System.out.println(super.a);

    SuperDemo td = new SuperDemo();

    td.m1();

}

}

```

BLOCK	METHOD
<ul style="list-style-type: none"> * Doesn't have identity * Automaticaly executes * We cannot call explicitly * Doesn't have return type * we can write return without value stat.. but we cannot write return with value. 	<ul style="list-style-type: none"> * Have an identity * Not executes automatically we need to call * We can call * Method must be have return type. * we can both place in method

Methods:

In java every value is hold by the variables.

If we want to update the value of an variables, we need some logic or operations.

In java, we can't do the operations in class level or we can't write logic in the class level.

So if we want to do the operations in java we have the following areas.

1. static Block.
2. Non-static block.
3. Constructors.
4. Methods.

If we are writing logic or operation in static block, that static block logic can be executed only one time, if we want to execute more than one we can't call explicitly.

If we are writing logic or operation in non-static block and constructor, those blocks execution need object creation, if we are executing those blocks multiple times, programmer should create multiple objects.

If we are keep on creating objects then we are unnecessarily wasting our memory.

To avoiding above drawbacks we have one special block in class level that is method.

We can execute the method multiple times based on our requirement.

The difference between block and method is blocks automatically executed whereas methods not executed, we need to call explicitly. Blocks don't have any specific name to call, whereas method have specific name to call.

Methods are mainly design for carrying the information from one place to another place and also used to holding the logic.

Ex:

```
class A{  
    System.out.println("not valid");//invalid  
    static {  
        s.o.p("only one time executed");  
    }  
    {
```

```
s.o.p("object creation mandatory");
}

A(){
    s.o.p("object creation mandatory");
}

void m1(){
    s.o.p("writing the logic in method very flexible");
}

}
```

Note: No method will be executed automatically, except main method.

Each and every class having its own functionalities, we can achieve these functionalities through methods.

```
class Account{
    double accountNo=123456789;
    String accHoldName="bhuvana";
    double amout=10000;
}

class Deposit{
    public void deposit(int amount){
        Account acc = new Account();
        acc.amout = acc.amout+amount;
    }
}

class Withdrawl{
```

```
public void withdrawl(int amount){  
    Account acc = new Account();  
    acc.amount=acc.amount-amount;  
}  
}  
  
class Transaction{  
    psvm(--){  
        Deposit d = new Deposit();  
        d.deposit(10000);  
        Withdrawl w = new Withdrawl();  
        w.withdrawl(5000);  
    }  
}
```

Note:

In the above program every class has its own functionalities.

We can differentiate with the help of different methods.

According to above information we can specify “method” is a block in the class, which contains set of statements, mostly we can say about these statements are called as actions or behavior or operation of a class.

All these statement have its own logic according to class specification.

That means each and every class has its own action and behavior.

For example sports class does not contain logic of Faculty class.

So the conclusion is, the logic of statements, which are located in the sports class method is different from logic of statements, which are located in the Faculty class methods.

The logic of a statement must be written in a method, not outside of a method.

If we attempt to write logic outside of method, the compiler will display the error message called as "identifier expected" System.out.println("hi");

Getting Modularity (differentiate from one logic to another logic).

We are always writing one logic within the one method.

Reduce Time Consuming.

Note: Java doesn't allow nested methods.

Mandatory things to communicate methods:

- >AccessModifier (private,default,protected,public)
- >Type of method(static/nonstatic)
- >MethodName
- > Parameter type and list

blocks are automatically executed whereas methods are not executed (we need to call).

blocks doesnot have identity whereas methods have identity.

by using identity we can able to call methods explicitly whereas block doesnot have identity, so we are unable to call explicitly.

blocks are executed the way we mention in the program from top to bottom, where as method are executed the way(order) we calling the methods.

Methods/Operations/Behaviour/Member methods:

Method is sub block of class.

Method is holding the logic, which doing some operations on properties

one object is communicating with another object with the help of methods.

Methods are having identity, which help of identity we can differentiate from one method to another method.

If we writing logic in static block we can execute only one time but we cannot multiple times, with the help of instance block and constructor, we can execute the logic multiple times, but we required more object (memory).

To avoiding above drawback, we are always prefer to writing the logic within methods.

classification of methods:

based on modifier:

static methods

non-static/instance methods

based on the body:

abstract methods

concrete methods

based on return type:

void methods

non-void methods

static void parameter

ns nv nonparam

s-v-p ns-v-p

s-v-np ns-v-np

s-nv-p ns-nv-p

s-nv-np ns-nv-np

zero argument/non-parameterized method

argument/parameterized method

based memory

factory methods

static factory methods

non-static factory methods

syntax:

=====

accessmodifier modifier modifier modifier modifier returntype

methodname(parameters){

list

}

method body

method signature ,

method prototype/heading

return type and method name and "()" are mandatory and remaining are not a mandatory

Types of methods:

Based on modifier:

- a. static methods
- b. non-static methods

Based on return type:

- a. void methods
- b. non-void methods

Based on parameters

- a. parameterized method
- b. non-parameterized method

Factory methods:

- a. static factory methods
- b. non-static factory methods

Based on body:

- a. abstract methods.
- b. concrete methods.

Based On modifier:

- a. static methods: A method which contains static keyword in its declaration or definition is called static method.

Ex:

```
static void m1(){  
}  
  
static int m2(){  
    return 100;  
}
```

```
static void m3(int x){  
}
```

static methods can be called by using the following ways.

1. By directly (method name)
2. By class name
3. by object or reference.

Note: Other class static method can be call by using two ways only

1. by class name
2. by object or reference

b. Non-static methods: A method which doesn't have static keyword in its declaration or definition is called non-static method.

Ex:

```
void m1(){  
}  
  
int m2(){  
    return 111;  
}  
  
void m3(int x){  
}
```

We can call non-static method by using two ways.

1. by directly
2. by object or reference

Note:

Other class non-static can be calls by using object or reference only.

```
class AA{  
    static void m3(){  
        System.out.println("AA class static m3 method");  
    }  
    void m4(){  
        System.out.println("AA class non-static m4 method");  
    }  
}  
  
public class MethodDemo {  
    static void m1(){  
        System.out.println("this is static m1 method");  
    }  
    void m2(){  
        System.out.println("this is non-static m2 method");  
        m5();  
    }  
    void m5(){  
        System.out.println("this is non-static m5 method");  
    }  
    public static void main(String[] args) {  
        m1();  
        MethodDemo.m1();  
        MethodDemo md = new MethodDemo();  
        md.m1();  
    }  
}
```

```
new MethodDemo().m1();

AA.m3();

AA obj = new AA();

obj.m3();

new AA().m3();

System.out.println("-----");

//m2();

md.m2();

obj.m4();

}

}
```

Note:

We cannot call non-static data directly from static area.(method/block)

Based on method return type:

a. void method:

A method, which doesn't carrying any information from one place to another place, is called void method.

Void methods are not calling from initialization phase.

void means nothing to define.(void means not zero, not false, not null simply nothing to define).

Within the void methods we can write return statement.

The return statement must be without value.

Note: void method can't be call from System.out.println().

```
void m1(){

}
```

```
s.o.p(m1());//invalid
```

Ex: void m1(){
 return ;
 //return 100;//invalid
}

void method: A method, which is not carrying any information from one place(called method) to another place(calling method) is called **void method**.

we declare void methods with the return type like "void".

void means nothing.

within the void method we can write return statement without value, but we cannot write return with any value.

void methods we cannot be call from s.o.p statement and from initialization place(after '=' operator).

b. Non-void method:

A method, which carrying the information is called non-void method.

Here information means

- a. primitive type
- b. primitive arraytype
- c. referenced type
- d. referenced array type

Every non-void method must be ended with return statement with respective value.

We can call non-void methods from s.o.p() statement.

Calling Method:

A method which is calling to another method, that method is called "Calling Method".

Called method: A method, which is called by other method, is called "Called Method".

Non-void methods:

A method which returns something otherthan void is called non-void or a method carrying some information from called method to calling method

*we can call directly
we can call in s.o.p
we can call from initialization place*

```
class NStudent{  
    String name="ramcharan";  
}  
  
public class MethodDemo {  
    void m1(){  
        System.out.println("non -static void m1()");  
        m2();  
        return;  
    }  
    static void m2(){  
        System.out.println("static void m2()");  
        //return 100;  
    }  
    int m3(){  
        System.out.println("m3 method");  
    }  
}
```

```
    return 100;
}

boolean m4(){
    System.out.println("m4 method");
    return false;
}

String m5(){
    System.out.println("m5 method");
    return "ram";
}

int[] m6(){
    System.out.println("m6 method");
    //int a[] ={10,20,30};
    //return a;
    int a =11;
    int b= 22;
    int c=33;
    //return new int[]{a,b,c};
    return new int[]{11,22,33};
}

NStudent m7(){
    System.out.println("m6 method");
    NStudent n = new NStudent();
    return n;
}
```

```
}

NStudent[] m8(){

    System.out.println("m6 method");

    NStudent n1 = new NStudent();

    NStudent n2 = new NStudent();

    NStudent n3 = new NStudent();

    NStudent n4[]={n1,n2,n3};

    //return n4;

    //return new NStudent[]{n1,n2,n3};

    return new NStudent[]{new NStudent(),new NStudent(),new NStudent()};

}

public static void main(String[] args) {

    m2();

    //System.out.println(m2());

    MethodDemo md = new MethodDemo();

    md.m1();

    //System.out.println(md.m1());

    System.out.println(md.m3());

    System.out.println(md.m4());

    System.out.println(md.m5());

    System.out.println(md.m6());

    int a[] = md.m6();

    for(int i=0;i<a.length;i++){

        System.out.println(a[i]);
    }
}
```

```

    }

    System.out.println(md.m7());

    NStudent ns = md.m7();

    System.out.println(ns.name);

    NStudent[] ns1 = md.m8();

    for(int i=0;i<ns1.length;i++){

        System.out.println(ns1[i].name);

    }

}

```

Based on Parameter:

a. zero argument method/non-parameterized method.

A method, which doesn't have any parameters in its parameter place is called non-parameterized method.

Ex:

```

void m1(  ){

}

int m2(  ){

    return 111;

}

```

b. Parameterized method:

A method, which contains parameters is called parameterized method.

The parameters may be

- a. primitive type
- b. primitive array type/reference
- c. referenced type
- d. referenced array type.

```
void m1(int x){  
}  
void m2(int x[]){  
}  
void m3(Student s){  
}  
void m4(Student s[]){  
}  
  
import java.util.Scanner;  
  
class MMM{  
    int x=999;  
}  
  
class PANP{  
    void m1( ){  
        System.out.println("non-parameterized m1() method");  
    }  
    void m2(int x){  
        System.out.println("param m2 method");  
        System.out.println(x);  
    }  
}
```

```
}

void m3(String x){
    System.out.println("param m3 method");
    System.out.println(x);
}

void m4(String x[]){
    System.out.println("param m4 method");
    for(int i=0;i<x.length;i++){
        System.out.println(x[i]);
    }
}

void m5(int x,String y){
    System.out.println("param m5 method");
    System.out.println(x);
    System.out.println(y);
}

void m6(MMM obj){
    System.out.println("m6 method");
    System.out.println(obj);
    System.out.println(obj.x);
}

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

```

PANP obj = new PANP();

MMM obj1 = new MMM ();

obj.m6(obj1);

obj.m6(new MMM());

obj.m1();

obj.m2(111);

obj.m3("abhinava");

String s[] = {"ram","java","php","oracle"};

obj.m4(s);

String s1[] = new String[]{"cobol","naresh i","techonologies"};

obj.m4(s1);

obj.m4(new String[]{"maths","english","telugu"});

String s2[] = new String[3];

Scanner scan = new Scanner(System.in);

System.out.println("enter some string values");

for(int i=0;i<s2.length;i++){

    s2[i]= scan.next();

}

obj.m4(s2);

obj.m5(111,"basha");

}

}

```

Abstract Method:

A method, which does not having body is called abstract method.

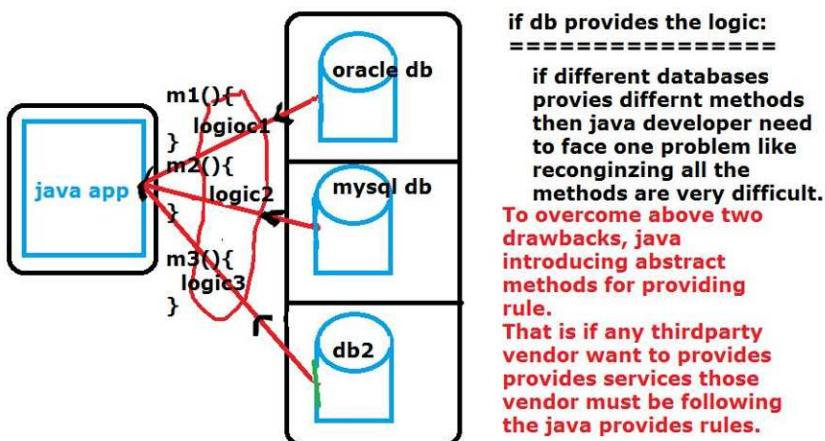
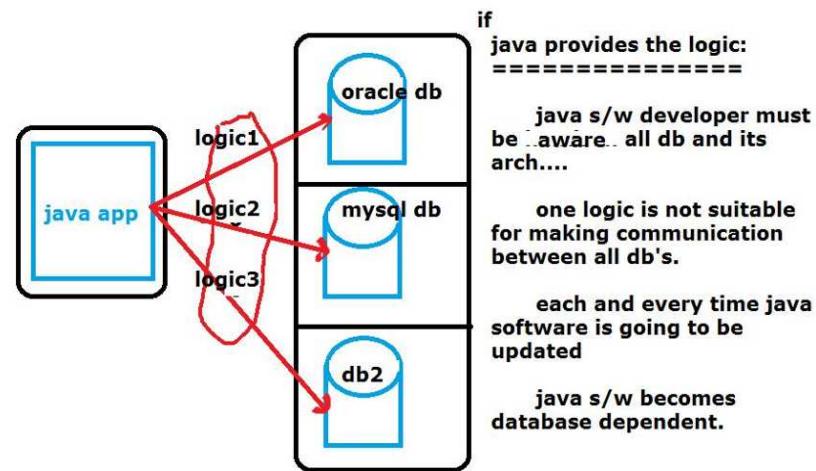
Abstract method is always ended with semicolon (;).

Abstract method must be needs "abstract" keyword in its declaration.

Abstract method always provides specification/rule.

If we want to use those methods we need to provide implementation for specification.

Ex: abstract void m1();



Concrete method:

A method which has body is called concrete method.

Ex:

```
void m1(){}
```

```
}
```

Data Hiding:

The data which is not accessible from outside of the class is called data hiding.

(or)

Doesn't provide any permissions to unauthorized user to interact with our secure data is called Data Hiding.

We can achieve data hiding concept with the help of private access modifier.

```
class A{  
    private int i=111;  
    private int j = 222;  
}
```

Data abstraction:

Hiding unnecessary information from enduser and provides only necessary information is called data abstraction.

Hiding the implementation logic from user to another user is called data abstraction.

We can achieve this data abstraction with the help of abstract keyword.

```
class A{  
    private int i;  
    public setI(int i){  
        this.i = i;  
    }  
}
```

```
}
```

```
class X{           class Y{  
    psvm(){          psvm(){  
        A obj = new A();      A obj = new A();  
        obj.setI(111);       obj.setI(222);  
    }                  }  
}                   }
```

In above syntax setI() will hide the both X,Y class logics.

JavaBean: It is a pojo, maintaining some standards.

those are:

1. class must be public.
2. class must implements java.io.Serializable or
java.io.Externalizable.
3. public zero argument constructor
4. Properties must be private
5. public setters and getters

POJO: is a normal java class.

note: Every javabean is pojo but pojo is not a java bean.

Example on Java Bean:

```
public class Account implements java.io.Serializable/Externalizable{  
    public Account(){  
    }  
  
    private long accountNo;  
    private short pin;  
  
    public void setAccountNo(long accountNo){  
        this.accountNo = accountNo;  
    }  
  
    public long getAccountNo(){  
        return accountNo;  
    }  
    // setters and getters for pin also  
}
```

Encapsulation:

Combination of providing services and security is called encapsulation.

Combination of private variable and public methods is called encapsulation.

.

Wrapping of data and its related methods is called encapsulation.

Placing the object information into a class is called encapsulation.

Hiding the information with the help of private modifier and accessing that hiding information with the help of public methods from outside of the program is called encapsulation.

Private data we can access outside of the class by using public method and reflection api.

```
class Bank{  
    private long accountNo;  
    private String accountHoldName;  
    private double amount;  
    public void setAccountNo(long accountNo) {  
        this.accountNo = accountNo;  
    }  
    public long getAccountNo() {  
        return accountNo;  
    }  
    public void setAccountHoldName(String accountHoldName) {  
        this.accountHoldName = accountHoldName;  
    }  
    public String getAccountHoldName() {
```

```
        return accountHoldName;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }

}

public class EncapsulationDemo {

    public static void main(String[] args) {
        Bank obj = new Bank();
        obj.setAccountNo(35568987);
        obj.setAccountHoldName("Srinivas");
        obj.setAmount(5000);
        long accountNo = obj.getAccountNo();
        String accountHoldName = obj.getAccountHoldName();
        double amount = obj.getAmount();

        System.out.println(accountNo+ " " +accountHoldName+ " " +amount);
    }
}

package oops;

import java.util.Scanner;

class Account{
```

```
private long accountNo;  
private String accountHolderName;  
private short pin;  
private double amount;  
public long getAccountNo() {  
    return accountNo;  
}  
public void setAccountNo(long accountNo) {  
    this.accountNo = accountNo;  
}  
public String getAccountHolderName() {  
    return accountHolderName;  
}  
public void setAccountHolderName(String accountHolderName) {  
    this.accountHolderName = accountHolderName;  
}  
public short getPin() {  
    return pin;  
}  
public void setPin(short pin) {  
    this.pin = pin;  
}  
public double getAmount() {  
    return amount;
```

```
}

public void setAmount(double amount) {
    this.amount = amount;
}

}

public class Encapsulation {
    static Scanner scan = new Scanner(System.in );
    public static void main(String[] args) {
        System.out.println("WELCOME TO SBI BANK");
        System.out.println("enter your pin to interact with your data");
        short pin = scan.nextShort();
        Account acc = new Account();
        if(pin == 5555){
            System.out.println("you are valid user ");
            acc.setAccountHolderName("RAMCHANDRA");
            System.out.println("what type of service do you want?");
            System.out.println("1.withdrawl\t 2.deposit");
            System.out.println("3.balance\t 4.accholdername");
            System.out.println("do want to continue if yes click on Y if No click N");
            String service = scan.next();
            boolean process = true;
            while(process){
                if(service.contains("y")){
                    System.out.println("choose ur service number");
                }
            }
        }
    }
}
```

```
byte serviceType = scan.nextByte();

switch(serviceType){

case 1: System.out.println("you are choosing withdrawl enter your
amount:");

    double amount = scan.nextDouble();

    if(acc.getAmount() >= amount){

        double amount1 = acc.getAmount()-amount;

        System.out.println("your balance is:"+amount1);

        acc.setAmount(amount1);

    }

    else

        System.out.println("you dont have sufficient amount");

    break;

case 2: System.out.println("you are choosing deposit enter your amount:");

    double amount1 = scan.nextDouble();

    double amount2 = acc.getAmount()+amount1;

    acc.setAmount(amount2);

    System.out.println("your balance is: "+acc.getAmount());

    break;

case 3: System.out.println("you are choosing balance");

    System.out.println("your balance is: "+acc.getAmount());

    break;

case 4: System.out.println("you are choosing account holdername: ");

    System.out.println("Account Holder name is:
"+acc.getAccountHolderName());
```

```
        break;

    }

}

else {

    System.out.println("THANK YOU FOR VISITING SBI
SITE");

}

System.out.println("do you want to continue if yes
click 'y' or if no click 'n'");

String ss = scan.next();

if(ss.contains("y"))

    process=true;

else

    process=false;

}

}

System.out.println("THANK YOU FOR VISITING SBI SITE");

}

}

class A{

public static void main(String...ram){

    B obj = new B();

    System.out.println(obj.a);

}

}
```

FileName: A.java

Steps to load the B's class information:

--> Compiler will check B .class information in A.java file. If not available

--> Compiler will check B. class in current working directory. If not available

-->compiler will check B.java file in current working directory. If not available compiler will one error, that is cannot find symbol class B.

If B.java file is available in current working directly then B.java file converted into B. class file then that byte code will be used in A.java file..

If B. class file is available, then compiler will check B.java file available or not. If B.java file is not available only B. class available then compiler use B. class file information in A.java file.

If both B. class and B.java file are available then compiler will check both file creation time. If B. class file creation time is more than B.java file then compiler blindly use B. class file.

Otherwise means B.java file creation time is more than the B. class file creation time then compiler first compile B.java file and updated the existed B. class with new B. class file information, later compiler use updated B. class file information.

Singleton Design pattern:

To stop to create multiple object for a particular class is called singleton design pattern.

With the help of singleton design patter we can save the memory.

Ex: Servlet.

```
class Bank{  
    private static Bank b = new Bank();  
    private Bank(){  
        System.out.println("Bank constructor");  
    }  
}
```

```
public static Bank getObject(){
    return b;
}

}

public class SingletonDemo {

    public static void main(String[] args) {
        //Bank b = new Bank();
        Bank obj = Bank.getObject();
        System.out.println(obj);
        Bank obj1 = Bank.getObject();
        System.out.println(obj1);
        Bank obj2 = Bank.getObject();
        System.out.println(obj2);
    }
}

class Account{
    Account(int x){
        System.out.println("account const");
        System.out.println(x);
    }
}

class Deposit extends Account{
```

```
Deposit(){  
    super(100);  
    System.out.println("Deposit const");  
}  
}  
  
public class EncapsulationDemo {  
    public static void main(String[] args) {  
        Deposit d = new Deposit();  
    }  
}
```

Note:

If super class having parameterized constructor, programmer must be write one constructor in subclass, and programmer must be write super (---) with parameter in that constructor.

Each every class is always holds its own constructor not other class constructor.

```
class A{  
    A(){  
    }  
    /*  
     * B (){//invalid  
     }  
     */  
}  
  
class B{
```

```
}
```

Factory Method:

A method which returns same class object or some other class object is called factory method.

Factory methods are two types

1. static factory method
2. non-static factory method

```
package constructor;

class Parent{

    String parentName = "KrishnaRao";

}

class Child{

    String childName ="RamaChandraRao";

    static Child getChild(){

        return new Child();

    }

    Parent getParent(){

        return new Parent();

    }

}

public class FactoryMethodDemo {

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

```
    Child child = Child getChild();
    Parent parent = child getParent();
    System.out.println(child);
    System.out.println(child.childName);
    System.out.println(parent);
    System.out.println(parent.parentName);
}
}
```

Interface:

A java element which provides 100% abstraction is called interface.

or

If we don't know any implementation then we can go for interface.

Ex:

```
interface I{
    public abstract void m1();
    public abstract void m2();
    public abstract void m3();
}
```

Abstract Class:

A java element which provides some part of abstraction is called abstract class.

If we know some part of implementation then we can go for abstract class.

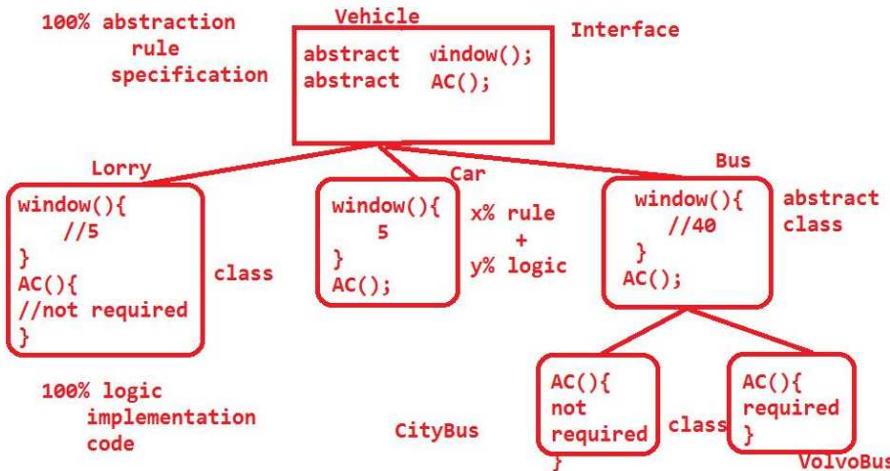
Ex:

```
abstract class AC{  
    public abstract void m1();  
    public abstract void m2();  
    public void m3(){  
        -----//logic  
    }  
}
```

Class:

A java element which provides 100% implementation is called class.

```
class Student{  
    public void m1(){  
        -----//logic  
    }  
    public void m2(){  
        -----//logic  
    }  
    public void m3(){  
        -----//logic  
    }  
}
```



default, private, public , protected, abstract, synchronized, final, static, strictfp, native.

The above are applied on methods

Bellow keywords are not applied on methods.

volatile

transient

Parameter:

A variable which is available at method parenthesis to hold value is called parameter.

Argument:

The value which is send to parameter is called argument.

```

class A{
    static void m1(int x){
        //x is a parameter
    }
    psvm(-){
        A obj = new A();
    }
}

```

```
    obj.m1(111);//111 is a argument  
}
```

Varargs:

Var args stands for variable argument.

It is used to hold zero or one or more values.

It is introduced in java 1.5 version

It can be represents with the help of "..." (3dot operators)

Don't use more than 3 dots.

Don't use less than 3 dots.

These 3 dots, we can be called as elipse.

Don't give any space between dots.

Valid syntax:

String ... x;

String... x;

String ...x;

Invalid syntax:

String . . . x;

String x;

String .. x;

String . ..x;

String.. .x;

String x...;

...String x;

If we are using the arrays, we can't be hold values more than it size;

```
int a[] = new int[5];
```

In the above syntax we can assign only 5 values, not more than 5 values.

If we are assigns less than 5 values, we have memory usage problem.

To overcome above two problems we can go for varargs.

Varargs must be use last parameter of a method, otherwise we will get compile time exception.

```
public class VarArgsDemo {  
  
    static void m1(int []a){  
  
        System.out.println("int array parameter m1 method");  
  
        for(int i=0;i<a.length;i++){  
  
            System.out.println(a[i]);  
  
        }  
  
    }  
  
    static void m2(int ... a){  
  
        System.out.println("-----");  
  
        System.out.println("m2 method");  
  
        for(int i=0;i<a.length;i++){  
  
            System.out.println(a[i]);  
  
        }  
  
    }  
  
    static void m3(int x, int ...y){  
  
        System.out.println("-----");  
  
        System.out.println(x);  
  
    }  
}
```

```
System.out.println("-----");
for(int i=0;i<y.length;i++){
    System.out.println(y[i]);
}
/*
static void m1(int...a){
}
static void m1(int a){
    System.out.println("int prameter m1 method");
    System.out.println(a);
}
/*static void m4(int...a,int x){
}
/*static void m5(int a,int...b,int c){
}
static void m1(){
    System.out.println(" zero argument m1 method");
}
public static void main(String[] args) {
    m1();
    m1(100);
    //m1(200)
```

```
m1(new int[]{11,22,33});  
m2();  
m2(111);  
m2(222,333);  
m2(444,555,666,777,888,999);  
m3(2323);  
m3(444,555);  
m3(666,777,888);  
//m4();  
//m4(111);  
m4(111,222);  
}  
}
```

Interface:

If we don't know any implementation or if we want to provide 100% abstraction then we can go for interface.

Interface is the communication channel between service provider and service consumer.

Interface is the combination of public static final variables and public abstract methods.

If we want to develop interface concept in java, we need to use one predefine keyword that is "interface". This is always followed by Interface_Name.

The naming conventions of interface is same as Class_Name.

All the variables are of interface must be initialized; otherwise we will get compile time error.

By default all the variables are public static final.

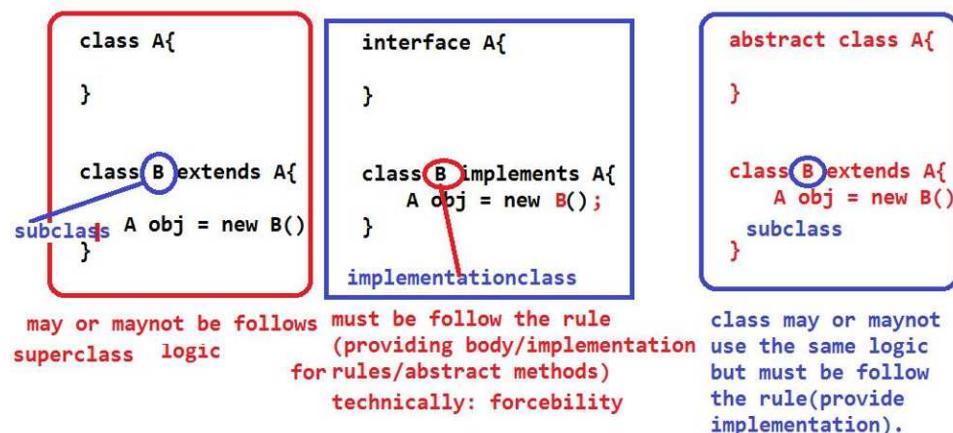
By default all the methods are public abstract.

static and abstract are invalid combination.

final and abstract are invalid combination. Interface provides commonality, forcibility.

If any class, abstract class want to communicate with interface we need one special keyword that is "implements" keyword.

We can write "abstract" keyword in front of interface keyword.



```
interface I{
```

```
    int i=111;
```

```
    void m1();
```

```
}
```

```
javac I.java
```

```
javap I
```

```
interface I{
```

```
public static final int a =111;  
public abstract void m1();  
}  
  
interface I{  
    int i=111;  
    final abstract void m1();//invalid  
    static abstract void m2();//invalid  
}
```

Based on, the methods of interface can be classified as three types from java 1.8 onwards.

- a. Marker interface
- b. Functional interface
- c. Normal interface.

Marker interface:

An interface which doesn't have any method is called marker interface.

Ex: `java.io.Serializable`, `java.lang.Cloneable`, `java.util.RandomAccess`.

If any class, which implements interface that class must be provide implementation for all the methods of interface. It is comes under forcibility.

If any class, which implements marker interface that class is treated as a special class by the JVM.

Functional Interface:

An interface, which contains only one method, is called Functional interface.

If we want to mention our interface as a functional interface, we should use one java 1.7 features, that `@FunctionalInterface` annotation.

Ex:

```
@FunctionalInterface  
interface I{  
    public abstract void m1();  
}
```

Normal interface:

An interface which contains more than one method or only one method without @FunctionalInterface annotation is called normal interface.

```
interface I{  
    public abstract void m1();  
    public abstract void m2();  
}  
  
interface J{  
    public abstract void m1();  
}
```

Types of interface:

1. General/Normal Interface A interface which contains one or more abstract methods

```
interface I{  
    public abstract void m1();  
}  
  
interface I{  
    public abstract void m1();  
    public abstract void m2();  
}
```
2. Marker Interface A interface which contains zero abstract methods

```
interface J{  
}  
ex: java.io.Serializable , java.lang.Cloneable,  
     java.util.RandomAccess
```
3. Functional Interface:(java 1.7)
A interface which contains only method and annotated with @FunctionalInterface.

```
@FunctionalInterface  
interface I{  public abstract void m1();  
}
```

Note:

We cannot create object for interface, but we can create reference. With the help of reference, we can hold its implementation class.

```
interface J{  
    public abstract void m1();  
}  
  
class M implements J {  
}
```

Here M is called implementation class.

We can write abstract keyword in front of interface keyword.

We can't write static and non-static blocks and constructors.

```
interface J{  
    static{//invalid  
}  
    {      // invalid  
}  
}  
  
interface K{  
    K(){//invalid  
}  
}
```

There is no super class for interface.

If we are trying to write concrete methods in java we are getting compile time error.

```
package interfaces;

interface I {    //extends java.lang.Object{

    int a=111;

    public abstract void m1();

    void m2();

    /*void m3(){

        System.out.println("concrete methods");

    }

    {

        System.out.println("non-static block");

    }

    static{

        System.out.println("static block");

    }

    I(){

        System.out.println("constructor");

    }*/}

}

class J implements I{

    public void m1(){

        System.out.println("J class m1 method");

    }

    public void m2(){

        System.out.println("J class m2 method");

    }

}
```

```
    }  
}  
  
public class InterfaceDemo {  
    public static void main(String[] args) {  
        //I obj = new I();  
        /*I obj ;  
        obj = new J();*/  
        I obj = new J();  
        obj.m1();//syntax (1)  
        obj.m2();//syntax (2)  
    }  
}
```

In above program syntax--1 compiler first check the type of obj, here type of obj is interface I. so first compiler will goes to I and check whether m1 method is available or not, if not compile time error, if yes m1() will bind to obj.

Then JVM will having the capability to check value.

JVM checks value, here value is J class memory(new J()).

So JVM executes the m1 method from J class.

Syntax--2 executions is also same as syntax---1.

**interface: 100% abstraction
specifications
(rules)** **all methods of interface must be
abstract.(upto 1.7)**

interface is communication channel between service provider and utilizer.

- *) No super class for interface
- *) we cannot write constructor, static and non static block and concrete methods
- *) by default variables are public static final and initialized.
- *) by default methods are public abstract
- *) we cannot create object
- *) we can create reference to hold its implementation class memory.
- *) we can write abstract keyword in front of the class.
- *) we can not abstract static, abstract private, abstract final methods.
- *) we can extract the one interface functionalities to another

Abstract Class:

If we want to provide some part of implementation or some part of abstraction then we can go for abstract class.

In the abstract class we can write both abstract and concrete method.

Class should be mention with "abstract" keyword.

Abstract class provides both forcibility and reusability.

Ex:

```
abstract class AC{  
    abstract void m1();  
    void m2(){  
    }  
}
```

We can't create object for abstract class

AC obj = new AC(); //invalid syntax:

But we can create reference.

```
AC obj;
```

With the help of abstract class reference, we can hold sub class implementation logic/memory.

If we want use functionalities of interface we need implementation class, in same manner if we want use the abstract class functionalities we need sub class.

The class which is extends abstract class is called sub class.

Ex:

```
class SAC extends AC{  
    void m1(){  
    }  
    void m2(){  
    }  
}
```

Note:

If any class which implements interface or extends abstract class, we should provide the body for all the method of interface and abstract class in implementation or sub class.

```
Interface I{  
    public abstract void m1();  
    public abstract void m2();  
}  
  
abstract class AC implements I{  
    public void m1(){}
    public abstract void m2();
    abstract void m3();
}
```

```
class SAC extends AC implements I{  
    public void m1(){  
    }  
    public void m2(){  
    }  
    void m3(){  
    }  
}
```

Within the abstract class we can write constructors.

Within the abstract class we can write both static and non-static blocks.

```
abstract class AC{  
    AC(){  
        S.o.p("constructor");  
    }  
    {  
        S.o.p("non-static block");  
    }  
    static{  
        S.o.p("static block");  
    }  
}
```

Abstract class is also subclass of java.lang.Object class.

```
abstract class AC{  
}
```

```
javac AC.java
```

After compilation compiler will convert in the following manner.

```
javap AC
```

```
abstract class AC extends java.lang.Object{  
    AC(){  
        super();  
    }  
}
```

Q) Why abstract class have constructor, why not interface?

interface doesn't need any predefine functionalities from java.lang.Object class, so need of forward the control from interface to Object class.

If we are not forwarding the control no need of constructor.

But whereas abstract class need some predefine functionalities of Object class, so we need to forward our control from abstract class to Object class. If we want to forward control from abstract class to Object class we need "super ()".

This "super ()" must be write in the constructor.

So for forwarding the control purpose we need constructor in abstract class.

Q) Why abstract class have static block why not interface?

A) All the variables of interface by default static final, so if we want to declare static final variables in interface level we need to initialize the static variables first. So already static variables are initialized by the programmer, so no need of using the static block again in interface.

But whereas static variables in abstract class may or may not be initialized. If not initializes, we can use static block for static variables initialization purpose.

Q) Why abstract class have non-static block why not interface?

A) Non-static blocks are used to initialize the non-static variables. But in interface every variable is static so no need of non-static block.

Whereas in abstract class, either we can write static and non-static variable, so if we want initializes the non-static variable, so we need non-static block.

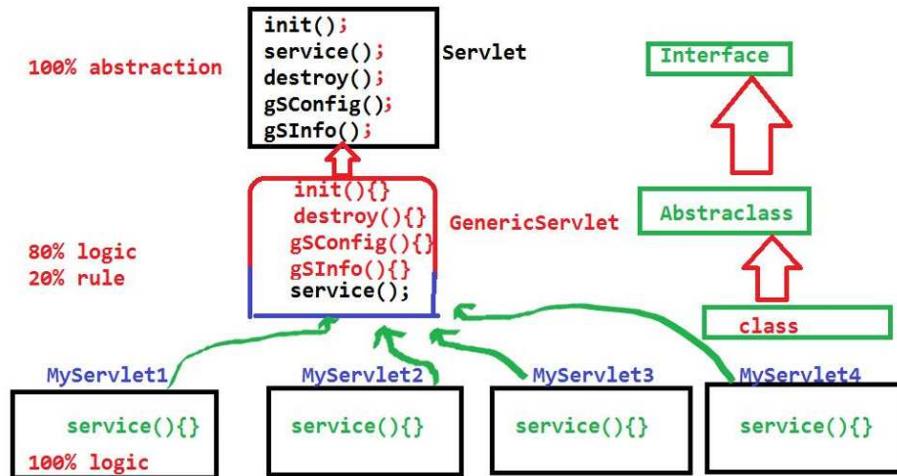
Q) Difference between interface and abstract class?**Interface:**

1. If we don't know any implementation then we can go for interface.
2. Interface provides 100% abstraction.
3. We cannot write protected, final, static, private, strictfp, native methods.
4. Every variable must public static final variables.
5. Interface variables don't have private, protected, transient, volatile keywords.
6. Interface variables must be initializes.
7. In the interface we can't write static, non-static blocks and constructors.
8. No super class for interfaces.

Abstract Class:

1. If we know some part of implementation then we go for abstract class.
2. Abstract class provides some part (percentage) of abstraction based on abstract methods.
3. We can write any type of method.
4. No need of public static final variable, if we want we can write.

5. We can write private, protected, transient, volatile keyword before variable names.
6. No need of initialization
7. We can write static, non-static blocks and constructor.
8. Abstract class is the sub class of java.lang.Object.



static block usage:

static block are used to initialize the static variables.

static block are used to registering the Driver class with DriverManager.

static block are used to provide common information.

Note: Abstract classes no need of having abstract methods.

abstract keyword is used to represent the non-body methods.

abstract keyword is used to provides some restrictions to end user to create an object for class.

abstract class AC{

```

void m1(){
}
  
```

```
void m2(){  
}  
}
```

If we want to use functionalities of abstract class we need subclass.

If any class which is not implements all the abstract methods of interface, we should be mentioning that class as an abstract class.

```
interface I{  
    public abstract void m1();  
    public abstract void m2();  
}
```

Invalid syntax:

```
class A implements I{  
    public void m2(){  
}  
}
```

Valid syntax:

```
abstract class A implements I{  
    public void m2(){  
}  
    public abstract void m1();  
}  
  
abstract class MM extends java.lang.Object{  
    MM(){
```

```
super();

System.out.println("MM class constructor");

}

void m1(){

    System.out.println("MM class m1 method");

}

void m2() {

    System.out.println("MM class m2 method");

}

abstract void m3();

}

class NN extends MM{

    NN(){

        super();

        System.out.println("NN constructor");

    }

    void m3(){

        System.out.println("NN class m3 method");

    }

}

public class AbstractDemo {

    public static void main(String[] args) {

        //MM obj = new MM();

        MM obj;
```

```
    obj = new NN();  
    obj.m1();  
    obj.m2();  
    obj.m3();  
}  
}
```

Note:

Relation between object and constructor:

Whenever we create object with the help of new keyword, we can execute the constructor.

Without object creation we can execute the constructor in abstract class.

Object is created but not constructor is executed.

B clone(), deserialization.

```
interface MM{  
    void m1();//public abstract void m1();  
    void m2();//public abstract void m2();  
    void m3();//public abstract void m3();  
}
```

```
abstract class NN implements MM{  
    public void m1(){  
        System.out.println("NN class m1 method");
```

```

}

public void m2(){

    System.out.println("NN class m2 method");

}

}

class OO extends NN{

    public void m3(){

        System.out.println("OO class m3 method");

    }

}

public class AbstractDemo {

    public static void main(String[] args) {

        OO obj = new OO();

        obj.m1();

        obj.m2();

        obj.m3();

    }

}

```

If any class which is not implements all the abstract methods of interface, we should be mentioning that class as an abstract class.

final means nobody inherit the functionalities.

By default constructors are not inherit, so declare constructor as final meaningless.

We cannot declare constructor as static.

static means class data, but constructor is purely related to object.

We cannot mention constructor as abstract, for this we have two reasons

The reason is whenever we create an object constructor automatically called so constructor must be having the body and also abstract means no body we should be use inheritance for providing body, but constructors are not participated in inheritance.

class	abstractclass	interface
• used to provides implementation/logic	• used to provides implementation and abstraction	• used to provides rules
• 100% logic/implementation	• some part of logic, and some part of rule	• 100% abstraction/rule (upto java 1.7)
• we have super classe for class	• we have super classe for abstract class	• we dont have super class
• we can write any type of variables and methods	• we can write any type of variables and methods	• we can write public static final variables and public abstract methods
• we can write blocks,constructors • we can create object • we can create reference • class can communicate with other class,ac,interface	• we can write blocks and constructors • we cannot create object • we can create reference • AC can communicate with class,AC,interface	• we can not write block and constructors • we can not create object • we can create reference • interface can communicate with another interface
class	abstractclass	interface
Variables can declare and initialized with the help of class reference variable we can hold same class memory or its subclass memory	variables can be declare and initialized with the help abstract calss we can hold only its sub class memory	variables must be initialized with the help interface reference we can hold its implementation class memory

Q) How can we call constructor of abstract class in java?

A) By creating object of it subclass.

```
abstract class A{  
    A(){  
        s.o.p("absract class");  
    }  
}  
class B extends A{  
    psvm(-){  
        new B();  
    }  
}
```

Q) Is the following syntax valid or not?

```
abstract class A{  
    A(int x){  
    }  
}  
class B extends A{  
}
```

Answer: NO

Q) how to resolve about problem?

We have two alternatives to resolve the problem.

1) Write default constructor and in B class and super class constructor, with the help of this syntax

```
B(){  
    super(111);  
}  
    (OR)
```

2) Write default constructor in class A.

```
class A{  
    A(int x){  
    }  
    A(){  
    }
```

Inheritance:

Extracting the functionalities from one class to another class is called inheritance without creating object. (static).

Without inheritance we need to face following problems.

1. Duplicate code/boilerplate of code.
2. Size of the code will be increasing
3. Compilation time will be increases.
4. Low performance
5. Development time will be increases.
6. Project cost will be increases.

To avoid above problems java uses one of the object oriented programming system principle that is "INHERITANCE".

Inheritance:

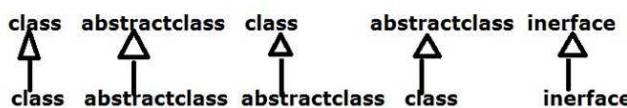
Extracting the functionalities from one class to another class is called inheritance.

We can develop inheritance in two ways

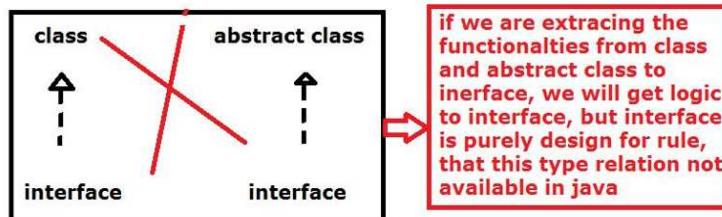
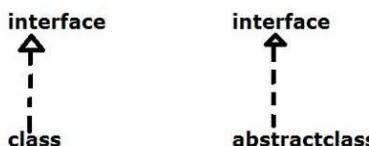
- 1.extends
- 2.implements



extends keyword we can use in the following ways:



implements



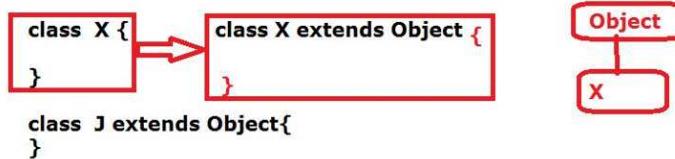
In java we have 5 types inheritances.

in java we have 5 types of inheritance

single inheritance
multilevel inheritance
hierarchical inheritance
multiple inheritance
hybrid inheritance

single inheritance:

writing a class alone or a class which having direct relation with object class is called single inheritance



a. Single Level Inheritance:

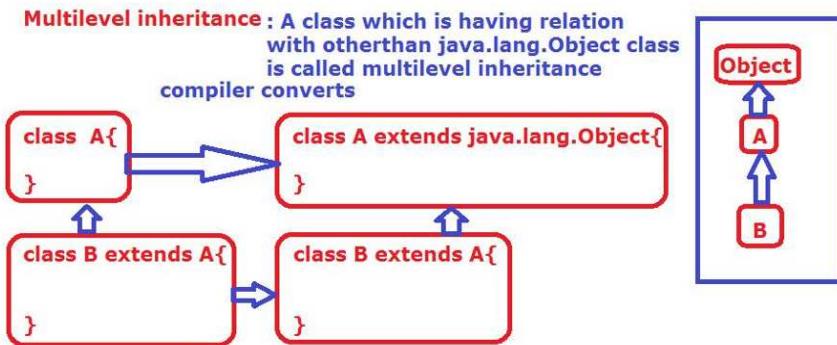
Extending the functionalities from `java.lang.Object` class is called single level inheritance.

```
class A extends java.lang.Object{  
}
```

b. Multi level inheritance:

Extends the functionalities from other than `java.lang.Object` class is called multilevel inheritance.

```
class A{  
}  
  
class B extends A{  
}
```



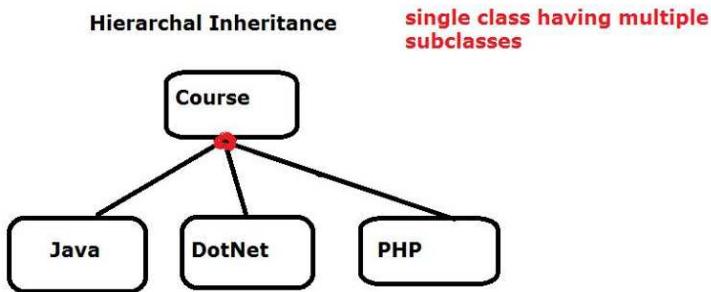
Note:

Every class is the subclass of `java.lang.Object` either directly or indirectly.

c. Hierarchical inheritance:

Derived the one class functionalities into more than one subclass is called hierachal inheritance.

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



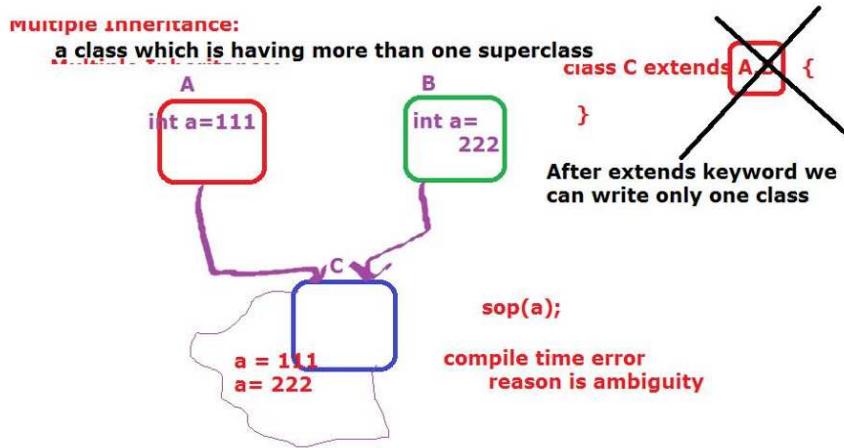
d. Multiple inheritance:

Extracting the functionalities from more than one super class to only one subclass is called multiple.

Java does not support multiple inheritances through classes. The reason is ambiguous problem.

```
class A{}  
}  
class B{}  
}  
class C extends A,B{}  
}
```

After extends keyword we can't write more than one class name.



e. Hybrid Inheritance:

Combination of all the inheritance is called hybrid inheritance. In hybrid inheritance also, we have multiple inheritance, so we can't implement hybrid inheritance in java.

Single Level + Multi Level

Single Level + Hierarchical

Single Level + Hierarchical + Multi Level

The above three combinations are comes under Hybrid. These type of combinations are, we can implement in java.

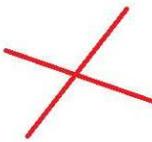
But if we add multiple inheritances to above combinations, then we cannot implement.

Hybrid inheritance:
the collection of any two inheritance is called Hybrid inheritance.

Single + HighLevel
Single + Hierachal
HighLevel + Hierachal
single+hierachal+highlevel



single+multiple
highlevel+multiple
hierachal + multiple



Note:

We can achieve multiple inheritances through interfaces. (only for methods not for variables).

We can develop inheritance with the help of two keywords.

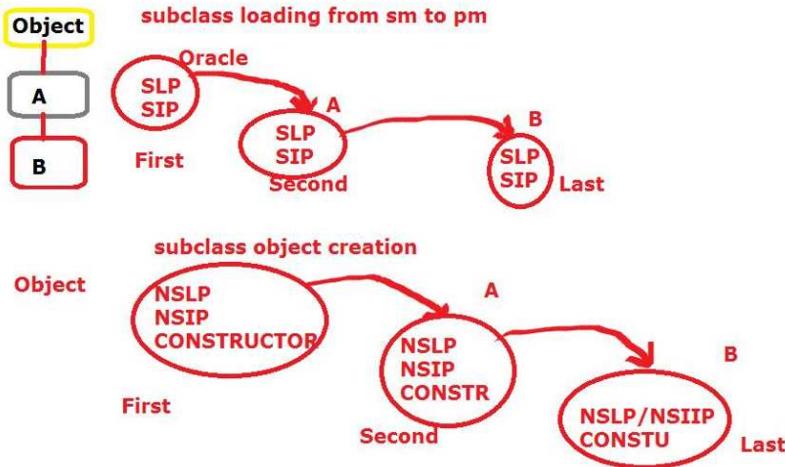
- a. extends
- b. implements

"extends" can be used in the following scenarios relation between

- a. class to class
- b. abstract class to abstract class
- c. interface to interface
- d. class to abstract class
- e. abstract class to class

"implements" can be used in the following scenarios relation between

- a. class to interface
- b. abstract class to interface



```

class A{
    int a = 10;
    A(){
        super();
        System.out.println("A class constructor");
    }
}

class B extends A{
    B(){
        super();
        System.out.println("B class constructor");
    }
    int b = 20;
}

public class InheritanceDemo {
    public static void main(String[] args) {
        B obj = new B();
    }
}

```

```
        System.out.println(obj.a);
        System.out.println(obj.b);
    }
}
```

abstract class to class:

```
class A{
    void m1(){
        System.out.println("A m1()");
    }
}

abstract class B extends A{
    abstract void m2();
}

class C extends B{
    void m2(){
        System.out.println("C m2() ");
    }
}

public class AbstractDemo1 {
    public static void main(String[] args) {
        C obj = new C();
        obj.m1();
        obj.m2();
    }
}
```

```
 }  
}
```

Note: If super class having parameterized constructor, in sub class programmer must be write one constructor, in that constructor we need to write one super(--) with matching argument syntax.

```
package inheritance;  
  
class A{  
  
    A(int x){  
  
        System.out.println("A class constructor");  
  
    }  
  
}  
  
class B extends A{  
  
    B(){  
  
        super(100);  
  
        System.out.println("B class constructor");  
  
    }  
  
}  
  
public class InheritanceDemo {  
  
    public static void main(String[] args) {  
  
        B obj = new B();  
  
        System.out.println(obj);  
  
    }  
  
}
```

Note: When ever Sub class is loading from secondary memory to primary memory.

First super class static variable and static block will be loaded after that sub class static variable and static blocks will be loaded.

When we create subclass object, then first super class non-static variables, non-static block, constructors are loaded after that sub class non-static variables, non-static blocks, constructors are loaded after that JVM will provides memory for subclass only.

Whenever subclass is loading, first super class static data is going to be loading and initialized later subclass static data is going to be loading and initialized later subclass main method will be executing.

whenever subclass object is creating super class non-static is going to be loading and initialized later super class constructor later sub class non-static data is going to be loading and initialized later subclass constructor executing.

once subclass constructor executing we can say object sucessfully initialized.

```
class A{  
    static int a = m1();  
    static int m1(){  
        System.out.println(a);  
        System.out.println("super class static m1 method");  
        return 111;  
    }  
    static{  
        System.out.println("super class static block");  
    }  
    int b = m2();  
    int m2(){
```

```
        System.out.println("-----");
        System.out.println(b);
        System.out.println("super class non-static m2 method");
        return 222;
    }
{
    System.out.println("super class non-static block");
}
A(){
    System.out.println("super class constructor");
}
}

class B extends A{
    static int c = m3();
    static int m3(){
        System.out.println("-----");
        System.out.println(c);
        System.out.println("sub class static m3 method");
        return 333;
    }
    static{
        System.out.println("sub class static block");
    }
    int d = m4();
```

```

int m4(){
    System.out.println("-----");
    System.out.println(d);
    System.out.println("sub class non-static m4 method");
    return 444;
}

B(){
    System.out.println("sub class constructor");
}

{
    System.out.println("sub non-static block");
}

}

public class InheritanceDemo {
    public static void main(String[] args) {
        B obj = new B();
        System.out.println(obj);
    }
}

```

Note:

Whenever we create object for sub class jvm only allocate the memory for subclass not for its super class.

```

class A{
    A(){}

```

```

        System.out.println("super class constructor");

    }

}

class B extends A{

    B(){

        System.out.println("sub class constructor");

        System.out.println(this.hashCode() +"..."+super.hashCode());

        System.out.println(this.getClass().getName()
        +"..."+super.getClass().getName());

    }

}

public class InheritanceDemo {

    public static void main(String[] args)throws InterruptedException {

        B obj= new B();

        Thread.sleep(365*1000*24*7);

        System.out.println("*****");

    }

}

```

Java Visual VM:

It is a tool, which is coming to our system mean while java software installation.

It is available in C:\Program Files\Java\Jdk\bin\jvisualvm.

With the help of this tool, we can recognize the number of instances created by JVM for our application.

Meanwhile of working with this jvisualvm tool our program must be in the running mode.

Steps to working Jvisualvm tool:

1. Go to C:\Program Files\Java\Jdk\bin folder.
 2. Search jvisualvm tool.
 3. Double click on jvisualvm
- Note:** program must be in running mode.
4. In the left side of tool, select our present running program.
 5. Right click on our program
 6. Click on open
 7. Click on monitor
 8. Click on heap dump
 9. Click on classes
10. Search our program (type our package name in the bottom of jvisualvm tool search box)

Example on multilevel and hierachal inheritance:

```
class A{  
    int a = 111;  
}  
  
class B extends A{  
    int b = 222;  
}  
  
class C extends B{  
    int c = 333;
```

```
}

class D extends A{

    int d = 444;

}

public class InheritanceDemo {

    public static void main(String[] args){

        C obj= new C();

        System.out.println(obj.a);

        System.out.println(obj.b);

        System.out.println(obj.c);

        D obj1 = new D();

        System.out.println(obj1.d);

        System.out.println(obj1.a);

    }

}
```

Ex:

```
class MM{

}

class NN extends MM{
```

Create object for subclass:

If we create object for subclass both sub and super class data can be access.

```
NN obj = new NN();
```

If we create an object for super class only super class data type can be access.

Up-casting or generalization or widening:

converting different type of objects to unique object.

why should go for upcasting?

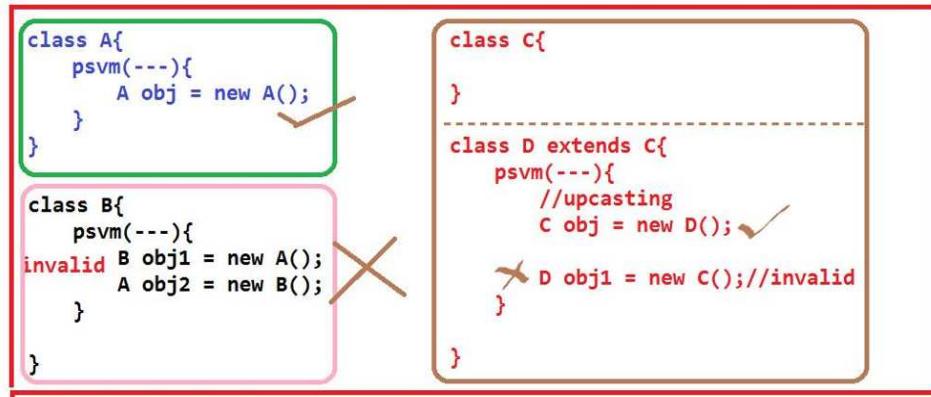
instead of writing individual logic for individual objects for , we should write single logic for different object and sending data from one place to another palcee.

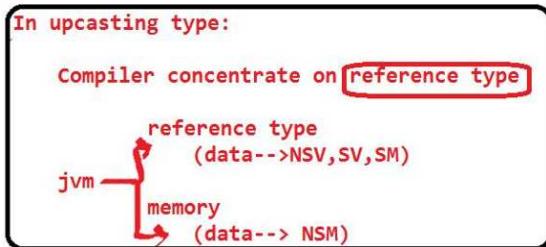
Placing sub class memory into super reference is called up-casting.

```
MM obj = new NN();
```

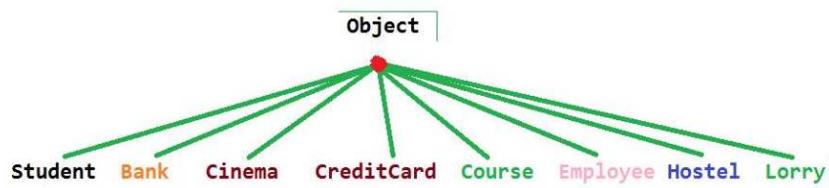
In the up-casting procedure, static variable, non-static variable, static methods are executed from reference type (super class and its super class).

Only non-static method are executed from memory (new NN ()--subclass and its super class).





In Normal object creation:
compiler concentrate on reference type
jvm concentrate on memory.



Syntax:

```

Object o = new Student();
Object o1 = new Bank();
Object o2 = new Cinema();
Object o3 = new CreditCard();
Object o4 = new Course();
Object o5 = new Employee();
Object o6 = new Hostel();
Object o7 = new Lorry();
  
```

```

class A{ }

class B{ }

class C extends A{ }

class UpCastingDemo{

    static public void main(String[] r){

        //A obj = new B(); //incompatible types

        //C obj1 = new A(); //incompatible types
  
```

```
/*A obj1 = new A();
C obj2 = (C) obj1;      java.lang.ClassCastException
*/
A obj1 = new C(); //Upcasting
C obj2 = (C)obj1; // Downcasting
}

}

class MM{
    static int a = 111;
    int b = 222;
    static void m3(){
        System.out.println("MM m3 method");
    }
    void m4(){
        System.out.println("NN m4 method");
    }
    void m1(){
        System.out.println("MM m1 method");
    }
}

class NN extends MM{
    static int a = 333;
    int b = 444;
    static void m3(){

```

```
        System.out.println("NN m3 method");
    }

void m4(){
    System.out.println("NN m4 method");
}

void m2(){
    System.out.println("NN m2 method");
}

}

public class InheritanceDemo {
    public static void main(String[] args){
        NN obj = new NN();
        obj.m1();
        obj.m2();
        System.out.println("-----");
        MM obj1 = new MM();
        obj1.m1();
        System.out.println("-----");
        MM obj2 = new NN(); //upcasting
        System.out.println(obj2.a);
        System.out.println(obj2.b);
        obj2.m3();
        obj2.m4();
        //NN obj3 = new MM();
    }
}
```

```
        System.out.println("-----");
        NN obj4 = (NN)obj2;
        System.out.println(obj4.a);
        System.out.println(obj4.b);
        obj4.m3();
        obj4.m4();
    }
}
```

Down-casting or specialization or narrowing:

Converting super class memory (subclass) into sub class type is called down-casting.

If we want do the down-casting first we need to up-casting.

```
MM obj = new NN(); //up-casting
NN obj1 = (NN) obj; //down-casting.
```

If we are creating object for subclass we will get both sub and super class data. mainly first preference gives to sub class later super class.

If we are creating object super class, we will get data from super class only not from sub class.

If we are creating an object like upcasting, then static,non-static variables, static methods are executing from super class, only non-static methods are executing from sub class. if non-static methods are not existed then executing from super class only.

In the above point data executing from super class only, if we want executing data from sub class then we will go to downcasting.

In java we have three types of relations.

1. Is-A relation

2. Has-A relation
3. Uses-A relation.

Is-A:

Accessing all functionalities from one class to another class, then we can go fro Is-a relation.

In java we can develop is a relation with the help of extends keyword.

```
class MM{}  
class NN extends MM{}
```

In the above relation we have a small drawback. That is if super class having 1000 variables, if we want to use some part of data (variables), then Is-A is not a best choice, the reason unnecessarily wasting of memory.

To avoid this drawback we can go for has-a relation.

Has-A:

Creating one class object within the another class is called has-a relation

```
class MM{}  
class NN{  
    psvm(){  
        MM obj = new MM();  
    }  
}
```

Uses-A: using one class reference in another class as a method parameter is called uses-a relation.

IS-A:

Making one class as a subclass of another class to extracting all the features of one class to another class is called IS-A relation.

```
Has-A: class B extends A{
    }
    Is-A
```

Creating one class object in the another class to extracting some part of the features, we called as Has-A

```
class A{
    int a = 111;
    int b = 222;
    int c = 333;
    void m1(){}
    void m2(){}
    void m3(){}
}

class B{
    psvm(-){
        A obj = new A(); Has-A
        sop(obj.b);
        obj.m3();
    }
}
```

This concept is comes under call by reference.

If we want use same memory data within the different location (method), then we need to forward that memory into different location(method) as argument.

```
abstract class AC{
```

```
    AC(int x){
```

```
}
```

```
}
```

```
class SAC extends AC{
```

```
}
```

In the above coding we will get compiler time error.

The reason is if we are not writing any constructor in SAC then compiler will provide the following code.

```
class SAC extends AC{
```

```
    SAC(){
```

```
        super();
```

```
}
```

```
}
```

super () will call super class (AC) zero argument constructor but there is no zero/default constructor in AC.

To overcome this problem we will follow the following remedies.

1. Write zero argument constructors in AC (super class).

or

2. Write constructor in SAC (subclass) and call argument constructor of AC (AC(int x){})

```
import java.util.Scanner;

interface Bill{
    double getRate();
    public double calculateBill(int units);
}

abstract class Plan implements Bill{
    double rate = 0;
    public abstract double getRate();
    public double calculateBill(int units){
        rate = getRate()*units;
        return rate;
    }
}

class DomesticPlan extends Plan{
    public double getRate(){
        return 3;
    }
}

class CommercialPlan extends Plan{
    public double getRate(){
        return 5;
    }
}

class PlanFactory{
    public Plan getPlan(String type){
        if(type == null){
            return null;
        }
        else if(type.equalsIgnoreCase("DomesticPlan")){
            return new DomesticPlan();
        }
        else if(type.equalsIgnoreCase("CommercialPlan")){
            return new CommercialPlan();
        }
        return null;
    }
}
```

```

        }
    }
public class ElectricityBill {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("enter your plan");
        String planName = scan.next();
        PlanFactory factory = new PlanFactory();
        Plan plan = factory.getPlan(planName);
        System.out.println("enter units");
        int units = scan.nextInt();
        if(plan !=null){
            double rate = plan.calculateBill(units);
            System.out.println("currentBill: "+rate);
        }
        else
            System.out.println("your plan type is not existed in the world");
    }
}

```

Polymorphism:

Polymorphism is one of the oops principle.

It is a mechanism to do same action within the different forms.

Polymorphism terminology comes from Greek language.

Poly means many and morphs means forms.

Polymorphism means many forms.

Polymorphism can classified into two types

- a. Static polymorphism or Compile time polymorphism.
- b. Dynamic polymorphism or Runtime polymorphism.

Polymorphism can be achieved through two techniques.

- a. method overloading.
- b. method overriding.

The same method can be existed in different forms is called polymorphism.

Polymorphism can achieve in two ways

1. Method Overloading
2. Method Overriding

```
class A{          MethodOverloading:  
    void m1(){  
    }  
    void m1(int x){  
    }  
    void m1(float x){  
    }  
    void m1(int x, float y){  
    }  
    void m1(float y, int x){  
    }  
}
```

Writing same method multiple times with in the same class with different parameters is called methodoverloading.

If we want to differentiate one method to another method, we need to concentrate on number of parameters, type of parameters, place or order of parameters.

Note: Method overloading never depends on parameter name, method return type.

Method Overloading:

Writing the same method with different parameters (Number of, type of, place of) with in the same class is called method overloading.

We cannot write same method with same parameters within the same class.

We should be differentiate one method to another method with the help of parameters.

Note: Don't consider variables names and return types.

1. Number of parameters.
2. Type of parameters.
3. Place of parameters.

Invalid syntax:

```
class A{  
    void m1(){  
    }  
    void m1(){//duplicate methods  
    }  
}
```

Valid syntax:

```
class A{  
    void m1(){  
    }  
    void m1(int x){  
    }  
    void m1(String x){  
    }  
    void m1(int x, String y){  
    }  
    void m1(String y, int x){  
    }  
}
```

Example program on Method Overloading:

```
public class MethodOverloadingDemo {  
    void m1(){  
        System.out.println("m1 method with zero argument");  
    }  
    void m1(int x){  
        System.out.println("m1 method with int argument");  
        System.out.println(x);  
    }  
    void m1(String x){  
        System.out.println("m1 method with string argument");  
        System.out.println(x);  
    }  
}
```

```
}

void m1(int x, float y){
    System.out.println("m1 method int-float argument");
    System.out.println(x);
    System.out.println(y);
}

void m1(float y, int x){
    System.out.println("m1 method float-int argument");
    System.out.println(y);
    System.out.println(x);
}

public static void main(String[] args) {
    MethodOverloadingDemo md = new MethodOverloadingDemo();
    md.m1();
    md.m1(234);
    md.m1("suji");
    //md.m1(10,20);
    md.m1(10,23.34f);
    md.m1(23.33f,10);
}

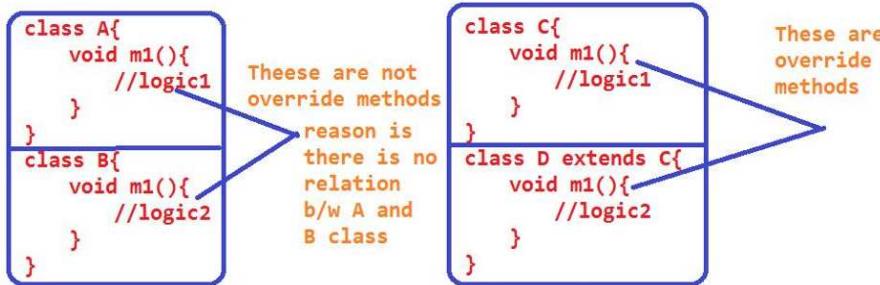
}
```

Method Overriding:

Whatever the logic, which is given by the java software or existed method is not suitable for our project requirement, then we should write our own logic within the existed method is called Method Overriding.

Method Overriding:

Whatever the logic, which is already existed or given by java software, if it is not supports to our project requirement then we should go for write our own project supporting logic with in the our class is called method overriding.



Method Overriding must be follows the following rules:

- 1) Access Modifier of a method in subclass must be same or increasable.

super class default method

sub class default, protected, public.

super class protected method

sub class protected, public

super class public method

sub class public

Rules for Method Overriding:

Subclass accessmodifier must be same or greater than super class accessmodifier.

super	default	protected	public
sub	default protected public	protected public	public

```

class A{
    void m1(){}
    protected void m2(){}
}

class B extends A{
    /*@Override
    private void m1(){
    }*/

    /*@Override
    void m1(){
    }*/

    /*@Override
    protected void m1(){
    }*/

    /*@Override
    public void m1(){
    }*/
}

```

2) Method return type must be same.

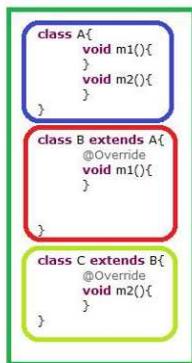
```
Method return type must be same in super and sub classes.  
class A{  
    void m1(){}
    float m2(){return 12.23f;}  
}  
  
class B extends A{  
    /*@Override
    void m1(){}
    */
    /*@Override //not a override
    int m1(){}
    /*int m2()//not a override
    return 111;
    */
}
```

Method overriding can not be consider the ranges.

MethodName must be same in super and sub classes.

```
class A{
    void m1(){}
}  
  
class B extends A{
    @Override
    void m2(){}//invalid
}
```

3) Method name must be same.



In C class m2 method is override method.
The reason is m2() is indirectly existed in B class, with the help inheritace concept.

B is subclass A, so A class m2() method will come B class.

4) Number of parameters, type of parameters and place of parameters must be same.(Method signature must be same)

Number of parameters, type of parameters, place of parameters must be same in super and sub classes.

```
class A{
    void m1(){}
    void m2(int x,float y){}
}
class B extends A{
    /*@Override
    void m1(int x){}//invalid
    */
    /*@Override
    void m2(int x,float y){
    }*/
    /*@Override//invalid
    void m2(float y,int x){
    }*/
}
```

5) throws clause exception class name can be same or we can delete throws clause or we can use throws clause with lower class(subclass) exception. But we cannot increase the exception level (superclass) and incompatible classes.

Super method having throws keyword, in the sub class we can use throws keyword with same class or its sub class or we can delete, but we can not write throws with incompatible class or its super class.

```
import java.io.FileNotFoundException;
import java.io.IOException;
class A{
    void m1()throws IOException{
    }
}
class B extends A{
    /*@Override
    void m1()throws IOException{
    }*/
    /*@Override
    void m1(){
    }*/
    /*@Override
    void m1()throws FileNotFoundException{
    }*/
    /* @Override //invalid
    void m1()throws CloneNotSupportedException{
    }*/
    /*@Override
    void m1()throws Exception{//invalid
    }*/
}
```

Note:

Method Overriding must be depends upon inheritance.

Variable name is not consider in the concept of method overloading and method overriding.

```
package inheritance;

import java.io.FileNotFoundException;
import java.io.IOException;

class L{

    void m1(){

    }

    void m2(int x, float y){

    }

    void m3() throws IOException{
```

```
    }

}

class M extends L{

    /*@Override
    void m3()throws IOException{
    }*/

    /* @Override
     * void m3(){
    }*/

    /*@Override
    void m3() throws FileNotFoundException{
    }*/

    /*@Override
    void m3()throws Exception{ //not override method
    }*/

    /*@Override
     * void m1(){
    }*/

    /*@Override
     * protected void m1(){
    }*/

    /*@Override
    public void m1(){
    }*/
}
```

```
/*@Override
public int m1(){//not override method
    return 100;
}*/
/*@Override
public void m2(){
}*/
/*@Override
void m2(int x){//not override method
}*/
/*@Override
void m2(int x, int y){ //not override method
}*/
/*@Override
void m2(float x, int y){//not override method
}*/
}
```

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

Note: private, static, final methods are not participated in the method overriding concept.

```
class M {
```

```
private void m1(){
    System.out.println("M class m1 method");
}

static void m2(){
    System.out.println("M class m2 method");
}

final void m3(){
    System.out.println("M class m3 method");
}

}

public class MethodOverridingDemo extends M {
    /*@Override
    private void m1(){
        System.out.println("MethodOverridingDemo class m1
                           method");
    }*/
    /*@Override
    static void m2(){
        System.out.println("MethodOverridingDemo class m2 method");
    }*/
    /*final void m3(){
        System.out.println("MethodOverridingDemo class m3 method");
    }*/
    public static void main(String[] args) {
```

```

MethodOverridingDemo md = new MethodOverridingDemo();

//md.m1();

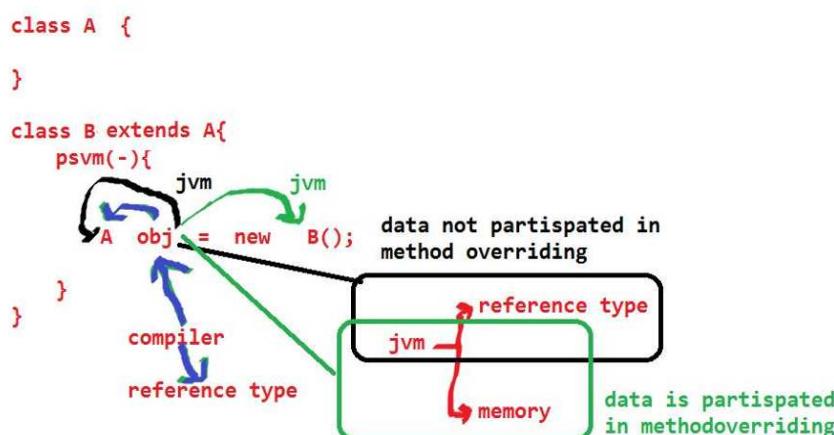
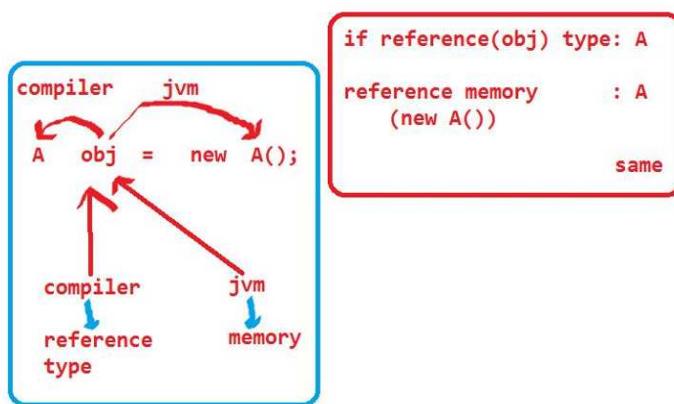
// md.m2();

}

}

```

If we are not using @Override annotation both private and static method are looks like participated in method overriding but they are not really participated in method overriding.



Static polymorphism: (Early binding)

A method which is bind by the compiler at compilation time, the same method is executed by the JVM at run time is called compile time polymorphism or static polymorphism.

Dynamic polymorphism: (lazy binding)

A method which is bind by the compiler at compilation time, the same method is not executed by the JVM at run time is called dynamic polymorphism or run time polymorphism or lazy binding.

CompileTimePolymorphisam:

The method which is bind by the compiler at compilation time, the same method executing by jvm at runtime is CTP.

only static method are partispated in CTP.

RunTimePolymorphisam:

the method which is bind by the compiler at compilation time, the same method not executing by the jvm at runtime(executing from subclass) is called RTP.

Non-static method are partispated in RTP.

MethodHiding:

Super class static method hides the execution of subclass static method is called method hiding.(static data/method never be partispated in overriding)

What type of method are not partispated in method overriding?
static,private,final methods are not partispated

```
package polymorphisam;
class A{
    static void m1(){
        System.out.println("super class static m1 method");
    }
    void m2(){
        System.out.println("super class non-static m2 method");
    }
}
class B extends A{
    //@Override
    static void m1(){
        System.out.println("sub class static m1 method");
    }
    void m2(){
        System.out.println("sub class non-static m2 method");
    }
}
public class PolymorphisamDemo {
    public static void main(String[] args) {
        A obj = new B();
        obj.m1();
        obj.m2();
    }
}
```

```
class MM{
    void m1(){
        System.out.println("MM m1 method");
    }
    void m2(){
        System.out.println("MM m2 method");
    }
    static void m3(){
        System.out.println("MM m3 method");
    }
}
```

```
class NN extends MM{
    void m2(){
        System.out.println("NN m2 method");
    }
}
```

```

    }

    static void m3(){

        System.out.println("NN m3 method");

    }

}

public class Polymorphisam {

    public static void main(String[] args) {

        NN obj = new NN();

        obj.m1();

        obj.m2();

        System.out.println("-----");

        MM obj1 = new NN(); //upcasting

        obj1.m1(); //CTP

        obj1.m2(); //RTP

        obj1.m3(); //CTP

    }

}

```

```

public class PolymorphisamDemo {
    void m1(int x){
        System.out.println("int argument method");
        System.out.println("x: "+x);
    }
    void m1(float x){
        System.out.println("float argument method");
        System.out.println("x: "+x);
    }
    public static void main(String[] args) {
        PolymorphisamDemo poly = new PolymorphisamDemo();
        poly.m1(111);
    }
}
Note: In the above compiler first preference will give to m1(int x)
method.
--> The reason are by default every non-fraction number will come to int type
--> Compiler first preference will give to lower range datatype.

```

```

public class PolymorphismDemo {
    void m1(Object x)
        System.out.println("Object argument method");
        System.out.println("x: "+x);
    }
    void m1(String x{
        System.out.println("String argument method");
        System.out.println("x: "+x);
    }
    public static void main(String[] args) {
        PolymorphismDemo poly = new PolymorphismDemo();
        String s = new String("ram");
        Object o = new String("sam");

        poly.m1(new String("yaane"));
        System.out.println("=====");
        poly.m1("varun");
    }
}
In the above program compiler/jvm will give first preference to
subclass later super class.

```

```

class A{
}
class B extends A{
}
class C extends B{
}
public class PolymorphismDemo {
    void m1(A x){
        System.out.println("A argument method");
        System.out.println("x: "+x);
    }
    void m1(B x){
        System.out.println("B argument method");
        System.out.println("x: "+x);
    }
    void m1(C x){
        System.out.println("C argument method");
        System.out.println("x: "+x);
    }
    void m1(Object x){
        System.out.println("Object argument method");
        System.out.println("x: "+x);
    }
    public static void main(String[] args) {
        PolymorphismDemo poly = new PolymorphismDemo();
        B obj = new B();
        poly.m1(obj);
        System.out.println("=====");
        poly.m1(new C());
    }
}

```

```

class Test{
    void m1(Object x){
        System.out.println("Object argument method");
        System.out.println("x: "+x);
    }
    /*void m1(String x){
        System.out.println("String argument method");
        System.out.println("x: "+x);
    }*/
    void m1(CharSequence x){
        System.out.println("CharSequence argument method");
        System.out.println("x: "+x);
    }
}

public class PolymorphisamDemo {
    public static void main(String[] args) {
        Test t = new Test();
        t.m1("ram");
    }
}

```

In this program first preference will give String later its interface (CharSequence) and finally gives control Object class



Method Hiding:

Super class static method, which is hiding the execution of sub class static method, is called method hiding.

JVM is always give the preference to subclass memory (new NN()).

If it is non-static method then JVM is executing from subclass, so there is no method hiding.

But if it is static method then JVM is not executing from subclass, it is executing from super class is called method hiding.

In the above class non-static m2 method is comes under method overriding, static m1 method is comes under method hiding.

Co-Variant return types:

This functionality introduced in java 1.5 versions.

In the method overriding concept the method return type must be same up to java 1.4.

But from java 1.5 onwards we can write subclass override method
return type can be subclass of super class method return type.

Note: Return type must be referenced type(non-primitive).

Ex:

```
class A{  
    A m1(){  
        new A();  
    }  
}  
  
class B extends A{  
    B m1(){//co-varient method  
        return new B();  
    }  
}
```

String Handling:

String is a collection of characters with in the double quotes.

Based on the character we can classify String into three types.

- 1) Numerical String
- 2) Character String
- 3) Special character String.

Numerical String:

If we are placing digits within the double quotes is called Numerical String.

Ex: String s = "123";

Character String:

If we are placing character with in the double quotes is called Character String.

Ex: String s = "jagadesh";

Special Characters String:

If we are placing any special characters with in the double quotes is called Special Character String.

Ex: String s = "\$%^&*"

From above information whatever the character we are writing within the double quotes that are comes under String.

We can write one or more character or combination of characters.(Digit, characters, special characters) within the double quotes.

String s = "R";

String s = "123abce%&*";

Java introducing String is an predefine class, which is available in java.lang package.

We can use String as a data type also.

Objects are classified into two types.

- 1) Immutable
- 2) Mutable.

Immutable Objects:

The value of an object has not changeable within the same memory; instead of changing it creates new object is called immutable.

Ex: All Wrapper classes (Byte, Short, Integer, Long, Float, Double, Character, Boolean), java.lang.String .

Mutable:

The value of an Object has been changed with in the memory is called mutable object.

Ex: java.lang.StringBuffer, java.lang. StringBuilder

If we want to check any two object in java we following ways.

- 1) By using “==” operator
 - 2) By using equals().
- 1) “==” operator is always checks hashCode of an objects.
 - 2) equals() of java.lang.Object class is also checks hashCode of an objects.

If we want to checks the content of objects then we can go for override the method within the String class. Java software developer has already override equals () in java.lang.String class.

The equals () of String class, is going to be check the content of an object.

In the java.lang.StringBuffer class equals() is not override.

Difference between

- 1) String s = "ram" and
- 2) String s1 = new String("ram");

In the first statement JVM will create only one object in String Constant Pool area.

In the second statement JVM will create two objects.

- a) One is in Heap Area
- b) Second is in String Constant Pool.

In the second statement reference (s1) is always points Heap Area object only.

Before creating object in String Constant Pool, JVM checks is there any object existed with "ram" content or not if not then it will create "ram" in SCP also. If object is existed JVM won't create any object in SCP, using the existed object only.

If any object is not having any external reference, those objects are easily garbage (de-allocate).

But in SCP object some time don't have any external reference, these are not garbage.

The objects which are available in SCP are strong objects.

String Constant Pool:

If we want to use the same objects repeated times, then place those Object's within the SCP only.

With the help of SCP we will get bellow advantages.

- 1) Memory utilization
- 2) Performance increase.

All the objects having same content then JVM is not create new object, JVM uses existed object memory only.

If anyone objects is doing updating, that updated (new) value is not affected to remaining objects, that updated value effected to only one object.

That JVM is not change the content of existed memory; instead it will create new memory with updated value.

SCP objects are very strong objects.

For each and every string we have memory in SCP.

intern (): If we want interact with the SCP data, then we have one method that intern().

```
public class CollectionDemo {  
    public static void main(String[] args){  
        String s = "ram";  
        System.out.println("s: "+System.identityHashCode(s));  
        s = "ram"+"chandra";  
        System.out.println("s: "+System.identityHashCode(s));  
        StringBuffer sb1 = new StringBuffer("ram");  
        System.out.println("sb1: "+System.identityHashCode(sb1));  
        sb1.append("chandra");  
        System.out.println("sb1: "+System.identityHashCode(sb1));  
    }  
}  
  
public class StringDemo {  
    public static void main(String[] args) {  
        String s1 = "ram";  
        String s2 = new String("ram");  
        System.out.println(s1==s2);  
        System.out.println(s1.equals(s2));  
        String s1 = "ram";  
        String s2 = "ram123";  
        s1.concat("123");  
    }  
}
```

```
System.out.println(s1);
s1=s1.concat("123");
System.out.println(s1);
System.out.println(s1==s2);
s1="ram"+"123";
System.out.println(s1==s2);

String s1 = "ram123";
String s2 = new String("ram");
System.out.println(s1==s2);
s2.concat("123");
System.out.println(s2);
s2=s2.concat("123");
System.out.println(s2);
System.out.println(s1==s2);
s2="ram"+"123";
System.out.println(s2==s1);
String s1 = "CoreJava";
String s2 = new String("AdvancedJava");
String s3 = "AdvancedJava";
String s4 = s2.intern();
System.out.println(s3==s4);
String s1 = "corejava";
String s2 = "COREJAVA";
```

```
String s3 = s1.toUpperCase();
System.out.println(s2==s3);

String s4 = s3.toUpperCase();
System.out.println(s3==s4);

System.out.println(s2==s4);

String s1 = "ram";
String s2 = "Ram";
System.out.println(s1==s2);
System.out.println(s1.equals(s2));
System.out.println(s1.equalsIgnoreCase(s2));

String s1 = new String ("java is an oopl");
String s2 = new String ("java is an oopl");
System.out.println(s1==s2);

String s3 = "java is an oopl";
System.out.println(s1==s3);

String s4 = "java is an oopl";
System.out.println(s3==s4);

String s5 = "java is "+"an oopl";
System.out.println(s4==s5);

String s6 = "java is ";
String s7 = s6+"an oopl";
System.out.println(s4==s7);

final String s8 = "java is ";
String s9 = s8+"an oopl";
```

```
        System.out.println(s4==s9);

    }

}
```

Note: If we are doing modification on string by using reference(either SCP reference or heap area reference) new object will be created in the heap area.

String s1 = "ram"

s1.concat("123");(new object is available in heap area)

If we are doing modification on string by using content then new object will be created in the SCP.

S1="ram"+"123";(new object is created in the scp).

Note: With the help of intern() we can interact with unreferenced SCP data.

```
public class StringDemo { //extends String {

    public static void main(String[] args) {

        String s = "internationalization";
        System.out.println(s.length());

        String s1="A";
        String s2 = "a";
        System.out.println(s1.compareTo(s2));
        System.out.println(s2.compareTo(s1));
        StringBuffer sb1 = new StringBuffer("a");
        StringBuffer sb2 = new StringBuffer("b");
        //System.out.println(sb1.compareTo(sb2));
        s=s.concat("--->java");
    }
}
```

```
System.out.println(s);
System.out.println(s1.equals(s2));
System.out.println(s.startsWith("inter"));
System.out.println(s.endsWith("inter"));
System.out.println(s1.equalsIgnoreCase(s2));
byte b[] = s1.getBytes();
System.out.println(b[0]);
int i = 100;
int j = 200;
String s3 = String.valueOf(i);
String s4 = String.valueOf(j);
String s5 = String.valueOf('d');
System.out.println(i+j);
System.out.println(s3+s4+s5);
System.out.println(s.lastIndexOf("n"));
System.out.println(s.lastIndexOf("nation"));
System.out.println(s.indexOf('t'));
System.out.println(s.indexOf('n'));
String s6 = "he";
System.out.println(s6.replace('e', 'i'));
System.out.println(s.replaceAll("nation", "india"));
String s7 = "java is an oopl";
String s8[] = s7.split(" ");
for(String s9: s8){
```

```

        System.out.println(s9);

    }

    String s10 = "10,20,30,40,50";
    int total = 0;
    String s11[] = s10.split(",");
    for(String s12: s11){

        System.out.println(s12);
        total = total+Integer.parseInt(s12);

    }

    System.out.println(total);

    //we cannot convert string value to character value
    //we dont have Character.parseChar().

}

}public class StringHandling {
    public static void main(String[] args) {
        String s1 = "ram chandra";
        System.out.println("Number of characters: "
                +s1.length());
        System.out.println("checking for data existed or not: "
                +s1.isEmpty());
        String s2 = "";
        System.out.println("checking for data existed or not: "
                +s2.isEmpty());
        //System.out.println("character at specific index position:
        "+s2.charAt(4));
        System.out.println("character at specific index position: "
                +s1.charAt(4));
        System.out.println("character ascii value at specific index position: "+
                s1.codePointAt(4));
        System.out.println("character ascii value at previous index" +
                " of specific index position: "+s1.codePointBefore(4));
    }
}
```

```

char[] c = new char[10];
String s3 = "internationalization";//20characters
s3.getChars(5,11,c,0);
for(int i=0;i<c.length;i++){
    System.out.println(c[i]);
}
System.out.println("=====");
byte[] b = new byte[10];
s3.getBytes(5,11,b,4);
for(int i=0;i<b.length;i++){
    System.out.println(b[i]);
}
System.out.println("=====");
b= s3.getBytes();
for(int i=0;i<b.length;i++){
    System.out.println(b[i]);
}
System.out.println("=====");
System.out.println("ram".equals("Ram"));
System.out.println("ram".equalsIgnoreCase("Ram"));

System.out.println("ram".compareTo("Ram"));//114-82
System.out.println("ram".compareTo("ramchandra"));//0-
number of characters(0-7==>-7)
System.out.println("ram".compareToIgnoreCase("Ram"));//114-
82
System.out.println(s3.startsWith("on"));
System.out.println(s3.startsWith("in"));
System.out.println(s3.endsWith("on"));
System.out.println(s3.endsWith("in"));
System.out.println("AB".hashCode());

System.out.println(s3.indexOf('n'));
System.out.println(s3.indexOf('n',11));
System.out.println(s3.indexOf("on"));
System.out.println(s3.indexOf("on", 10));

System.out.println(s3.lastIndexOf('n'));
System.out.println(s3.lastIndexOf("in"));
System.out.println("*****"+s3.lastIndexOf('n', 8));
System.out.println(s3.substring(5));
System.out.println(s3.substring(5, 11));

s1.concat(" nit");

```

```

        System.out.println(s1);
        s1= s1.concat(" nit");
        System.out.println(s1);

        System.out.println("nitn".replace('n', 'k'));
        System.out.println("ram chandra ram".replaceFirst("ram",
"java"));
        System.out.println("ram chandra ram".replaceAll("ram",
"java"));

        String s4 = "naresh i technologies";
        String[] s5 = s4.split(" ");
        for(int i=0;i<s5.length;i++){
            System.out.println(s5[i]);
        }
        char[] c3 = new char[10];
        c3 = "ram".toCharArray();
        for(int i=0;i<c3.length;i++){
            System.out.println(c3[i]);
        }
        String s6 = "    varun kirn ram sam yaane    ";
        System.out.println("s6: "+s6);
        String s7= s6.trim();
        System.out.println("s7: "+s7);

        System.out.println("ram".toUpperCase());
        System.out.println("VARUN".toLowerCase());
    }
}

```

Difference between String and StringBuilder and StringBuffer

String:

- Immutable
- Java 1.0
- Not synchronized
- Concat()
- compareTo()
- Not ThreadSafe
- Higher performance
- Not good at result
- Implements java.lang.Comparable

- No reverse()
- No capacity()

StringBuffer:

- mutable
- Java 1.0
- synchnoized
- append
- NO compareTo()
- ThreadSafe
- low performance
- good at result
- not Implements java.lang.Comparable
- reverse()
- capacity()

StringBuilder:

- mutable
- Java 1.5
- Not synchnoized
- append
- NO compareTo()
- Not ThreadSafe
- Higher performance
- Not good at result
- Not Implements java.lang.Comparable
- reverse()
- capacity()

	String	StringBuffer	StringBuilder
Version	1.0	1.0	1.5
ThreadSafe	No	Yes	No
Synchronized	No	Yes	No
Performance	Yes	No	Yes
append()	No	Yes	Yes
concat()	Yes	No	No
reverse()	No	Yes	Yes
implements Comparable	Yes	No	No
compareTo()	Yes	No	No
capacity()	No	Yes	Yes

```

public class StringHandling {

    public static void main(String[] args) {

        String s = "ram";

        String s1 = new String();

        System.out.println(s);

        System.out.println(s1);

        String s2 = new String("ramchandra");

        System.out.println(s2);

        String s3 = new String("");

        System.out.println(s3+"***");

        byte b[] = {65,66,67,68,69,70,71,72};

        String s4 = new String(b);

        System.out.println(s4);

        String s5 = new String(b,2,4);

        System.out.println(s5);

        char c[] = {'a','b','c','d','e','f'};

        String s6 = new String(c);
    }
}

```

```

        System.out.println(s6);

        String s7 = new String(c,1,3);

        System.out.println(s7);

        StringBuffer sb = new StringBuffer("standard");

        String s8 = new String(sb);

        System.out.println(s8);

        StringBuilder sb1 = new StringBuilder("Edition");

        String s9 = new String(sb1);

        System.out.println(s9);

    }

}

```

Note: If we want to use the compareTo() , both objects must be homogeneous and both objects must be implements java.lang.Comparable.

```

public class StringHandling {
    public static void main(String[] args) {
        String s1 = "internationalization";
        System.out.println("length: "+s1.length());
        System.out.println(s1.charAt(5));
        System.out.println(s1.codePointAt(5));
        System.out.println(s1.codePointBefore(5));
        System.out.println(s1.concat(" java"));
        System.out.println("ram".compareTo("Ram"));//r-R=114-82
        System.out.println("Ram".compareTo("ram"));
        System.out.println("Ram".compareTo("Ram"));
        System.out.println("Ram".compareTo("RamChandra"));
        System.out.println("ram".compareToIgnoreCase("Ram"));
        System.out.println(s1.endsWith("m"));
        System.out.println(s1.endsWith("on"));
        System.out.println("ram".equals("Ram"));
        System.out.println("ram".equalsIgnoreCase("Ram"));
        System.out.println(s1.indexOf('n'));
        System.out.println(s1.indexOf("nation"));
        System.out.println(s1.lastIndexOf('n'));
    }
}

```

```

System.out.println(s1.indexOf("on"));
System.out.println(s1.lastIndexOf("on"));
System.out.println(s1.indexOf('n',6));
System.out.println(s1.indexOf("on",10));
System.out.println(s1.isEmpty());
//String s2 = " ";
String s2 = "";
System.out.println(s2.isEmpty());
String s3 = "AB";
System.out.println(s3.hashCode());
System.out.println(System.identityHashCode(s3));

char a[] = new char[10];
String s4 = "nation";
a=s4.toCharArray();
for(char a1: a){
    System.out.println(a1);
}
System.out.println("VARUN".toLowerCase());
System.out.println(s4.toUpperCase());
String s5 = "    kit nit ramchandra java    ";
System.out.println("s5: "+s5);
String s6 = s5;
System.out.println("s6: "+s6);
String s7 = s5.trim();
System.out.println("s7: "+s7);
String s8 = "ram sam nit kit vikram";
String s9[] = s8.split(" ");
System.out.println(s8);
for(String s10: s9){
    System.out.println(s10);
}
String s10="naresh i technologies";
String s11 = s10.replace('i', 'I');
System.out.println(s11);
String s12 = s10.replaceAll("naresh", "karthik");
System.out.println(s12);

}
}

```

JVM ARCHITECTURE:

Virtual Machine:

VM is an software/Application which will provide environment or memory areas for executing the programs.

JVM: JVM is an software/application which will provide environment or five individual identical memory areas for executing the java programs.

We have two types of JVM

- 1) Client JVM
- 2) Server JVM.

JVM is only provides abstraction.

JRE is provides implementation for that abstraction.

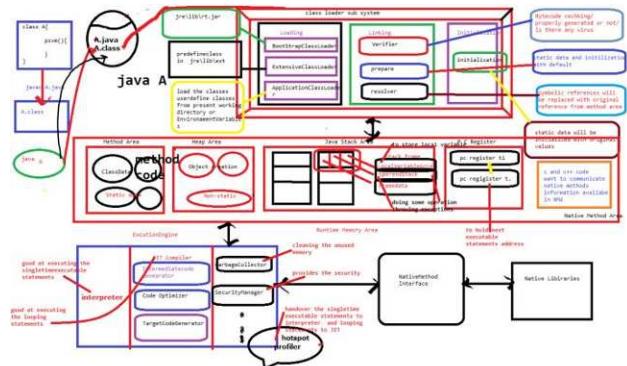
We have different JVM'S for different OS.

That means we have different JRE software for different Operating Systems.

If we want to execute the java program, first we should interact with the JVM.

With the help of "java" command we communicate with JVM. "java" keyword is always followed by class name.

syntax: `java Class_Name`



Java is internally communicate the with JVM, then JVM will come into the picture, first JVM will read class name and JVM control will go to secondary memory, that class byte code will be loaded from secondary memory to primary memory.

JVM internally uses ClassLoaderSubSystem to load the .class files from secondary memory to primary memory.

In the ClassLoaderSubSystem, we have three separate phases. Those are

- 1) Loading phase.
- 2) Linking phase.
- 3) Initialization phase.

Loading Phase: In this phase the byte code (predefine and user define class) will be loaded from secondary memory to primary with the help of bellow classloaders.

In this phase we have three ClassLoaderSubSystems.

1. ApplicationClassLoaderSubSystem
2. ExtensiveClassLoaderSubSystem
3. BootstrapClassLoaderSubSystem.

ApplicationClassLoaderSubSystem:

It will loads user define class files from current directly and environment variable path.

ExtensiveClassLoaderSubSystem:

It will loads predefine class files from jre\lib\ext folder.

BootstrapClassLoaderSubSystem:

It will load predefine class files from jre\lib\rt.jar file.

In the above three phases class is not available, then we will get one error cannot find or load main class <class_name>.

If available that .class file will loaded from secondary memory to primary memory and handover to linking phase.

Linking:

In this phase the loaded byte code will be checked by verifier.

In this phase we have three components.

1. Verifier
2. Preparer
3. Resolver.

Verifier will check whether the byte code is properly organized or not, is there any virus and hacking code or not. If yes verifier will give verifier error, if not that byte code will be handover to preparer.

Preparer will provide the default values to static variables in loading phase.

Resolver: it will convert symbolic reference into original references.

After this phase code will be handed over to initializer.

Initialization: all default values of static data will be replaced with original or actual data.

Runtime Memory Areas:

JVM provides 5 runtime memory areas.

- 1) Method Area
- 2) Heap Area
- 3) Java Stack Area
- 4) PC Registers
- 5) Native Method Area

Method Area: In this area all class data is stored. that means all the static data is available in the method area.

Heap Area: in this area all object data is stored. that means all the non-static data is available in heap area.

All the objects are created in the heap area only.

Java Stack Area: Every thread have its own stack.

These stacks will interact with local data/method level data. Java stacks have different slots, those slots are called stack frames.

Java Stack Frame will convert into three parts.

1) Local Variable Storage Area:

In this all local variables are stored.

2) Operand Stack:

in this phase all operations and calculations will be happens.

3) FrameData:

If method contains any exceptions, those exceptions will be thrown by frame data only.

PC Registers:

Every thread have its own PC Registers, these registers will hold the next execution statements address.

Native method area:

If java wants communicate with c and c++ code, that code will be available in Native method area.

If we interact with c and c++ code, we need libraries, those will be given by Native Library, that library will be handover by Native Method Interface to executable engine.

Execution Engine:

Internally JVM uses two translators to convert byte code to executable code.

1. Interpreter
2. JIT Compiler

Interpreter is good at execute the single time execution statements.

JIT compiler is good at execute the looping statements.

But these two are unable to find out behavior of statements.

In this one special component come into picture that is "profiler"

First profiler will identify the weather statements are single time or looping statements.

If single time execution statement then those statements handover to interpreter otherwise handover to JIT compiler.

Java is a high performance language, the reason java internally uses two translators to convert our byte code to executable. So java applications are executed within less time. If the application executes within less time automatically the performance will increase.

In this executable engine we have some other special components.

Those are Garbage Collector and Security manager.

Wrapper classes:

Representing the primitive data in the format of object is called wrapper class.

Wrapper class wraps the primitive data in the format of object.

When we create an object to a wrapper class, it contains a field/area and in this field, we can store a primitive data type.

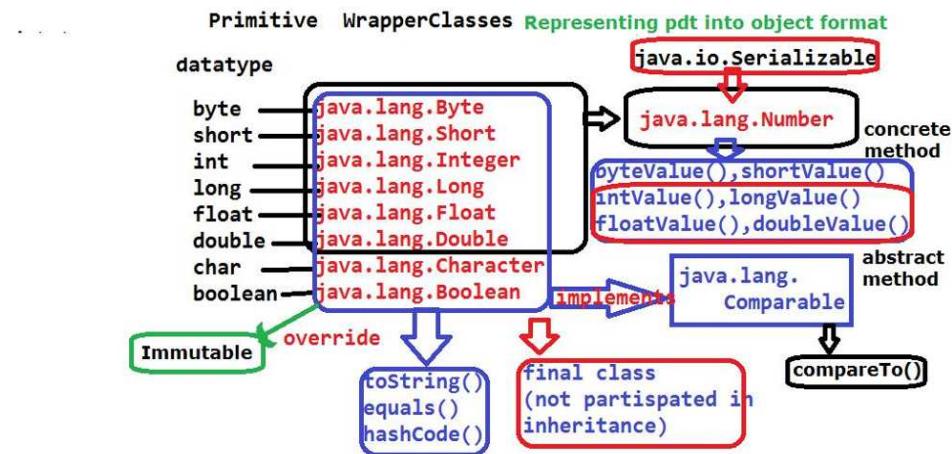
Whenever we send the data throughout the networks we should always prefer to send the data in the form of object.

If we want use any predefine logic (method), we cannot call on top of primitive variable, but we can call on top of object.

From above two conditions we need Wrapper class.

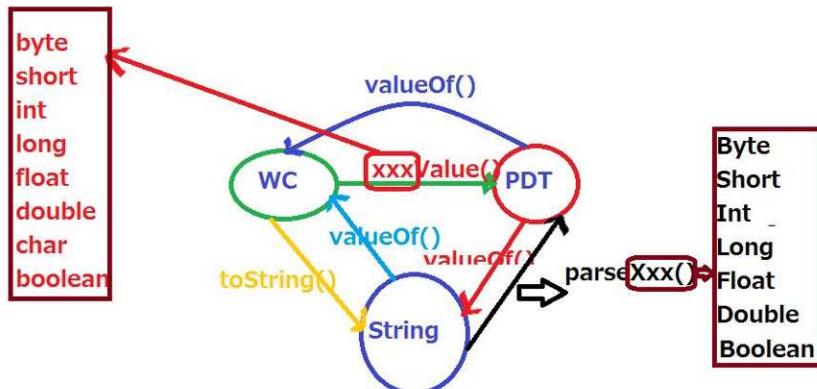
Primitive	wrapper class
byte	Byte

short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean



We can convert primitive data into wrapper class object in two ways.

1. By using constructors.
2. By using Methods.

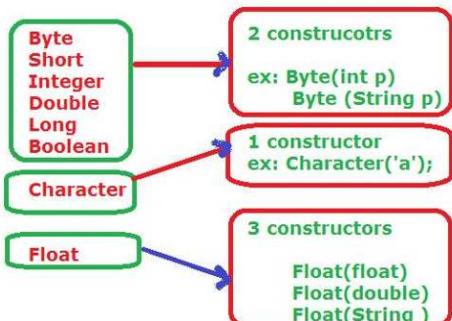


Note: String data we cannot convert into primitive char as well as wrapper class Character directly

Constructors in Wrappers class:

We can represent primitive data into the object by using following ways.

1. By Constructor
2. By Methods
3. By Autoboxing / Auto unboxing



1) java.lang.Byte:

This class contains two constructors.

Byte (byte value)--10

Byte (String value)"10"

2) java.lang.Short

This class contains two constructors.

Short (short value)

Short (String value)

3) java.lang.Integer

This class contains two constructors.

Integer (int value)

Integer (String value)

4) java.lang.Long

This class contains two Constructors

Long (long value)

Long (String value)

5) java.lang.Float

This class contains three constructors.

Float (float value)

Float (double value)

Float (String value)

6) java.lang.Double

This class contains two constructors.

Double (double value)

Double (String value)

7) java.lang.Character

This class contains only one constructor

Character (char value)

8) java.lang.Boolean

This class contains two constructors

Boolean (Boolean value)

Boolean (String value)

Java.lang.Number:

It is an abstract class.

It is the super class for Byte, Short, Integer, Long, Float, Double.

This class contains 4 abstract methods and two concrete methods.

The above 4 abstract methods are implemented in their sub classes (6)

It implements java.io.Serializable

Note:

All wrapper classes are implements Serializable and Comparable interfaces.

All the wrapper classes are immutable objects.

Note: All the wrapper class override the hashCode(), toString(), equals().

We can use '+' operator on top of wrapper class reference variables but not normal objects (references).

```
public class WrapperDemo {  
    public static void main(String[] args) {  
        byte b = 100;  
        Byte a1 = new Byte(b);  
        Byte a2 = new Byte("127");  
        System.out.println(a1);  
        System.out.println(a2);  
        Short a3 = new Short((short)120);  
        Integer a4 = new Integer(125);  
        Long a5 = new Long(120);  
        Float a6 = new Float("12.34f");  
        System.out.println(a6);  
        /*Long a7 = new Long("120l");
```

```
System.out.println(a7);*/
Double a8 = new Double(23.34d);
System.out.println(a8);
Double a9 = new Double("23.34d");
System.out.println(a9);
Character a10 = new Character('a');
System.out.println(a10);
Boolean a11 = new Boolean("false");
System.out.println(a11);
Boolean a12 = new Boolean("true1");
System.out.println(a12);

}

}
```

```
public class WrapperDemo {

    public static void main(String[] args) {
        //Primitive data type to Wrapperclass type
        Byte a1 = Byte.valueOf((byte)100);
        Short a2 = Short.valueOf((short)122);
        Integer a3 = Integer.valueOf(100);
        Long a4 = Long.valueOf(123l);
        Float a5 = Float.valueOf(12.34f);
        Double a6 = Double.valueOf(12.34);
```

```
Character a7 = Character.valueOf('a');

Boolean a8 = Boolean.valueOf(false);

System.out.println(a1+"...."+a2+"...."+a3+
"...."+a4+"...."+a5+"...."+a6+"...."+a7+"...."+a8);

//Wrapper class data to primitive datatype

byte b1 = a1.byteValue();

short b2 = a2.shortValue();

int b3 = a3.intValue();

long b4 = a4.longValue();

float b5 = a5.floatValue();

double b6 = a6.doubleValue();

char b7 = a7.charValue();

boolean b8 = a8.booleanValue();

System.out.println(b1+"...."+b2+"...."+b3+
"...."+b4+"...."+b5+"...."+b6+"..." +a7+"...."+a8);

//String data type to Wrapperclass type

Byte a1 = Byte.valueOf("100");

Short a2 = Short.valueOf("122");

Integer a3 = Integer.valueOf("100");

Long a4 = Long.valueOf("123l");//here l is not deleted

//Long a4 = Long.valueOf("123");

Float a5 = Float.valueOf("12.34f");

Double a6 = Double.valueOf("12.34d");
```

```
//Character a7 = Character.valueOf("a");

Boolean a8 = Boolean.valueOf("false");

System.out.println(a1+"...."+a2+"...."+a3+
"...."+a4+"...."+a5+"...."+a6+"...."+a8);

}

}
```

```
public class WrapperDemo {

    public static void main(String[] args) {

        //String data type to Wrapperclass type

        Byte a1 = Byte.valueOf("100");

        Short a2 = Short.valueOf("122");

        Integer a3 = Integer.valueOf("100");

        //Long a4 = Long.valueOf("123l");//here l is not deleted

        Long a4 = Long.valueOf("123");

        Float a5 = Float.valueOf("12.34f");

        Double a6 = Double.valueOf("12.34d");

        //Character a7 = Character.valueOf("a");

        Boolean a8 = Boolean.valueOf("false");

        System.out.println(a1+"...."+a2+"...."+a3+
"...."+a4+"...."+a5+"...."+a6+"...."+a8);

        //wrapper class to string
```

```
String b1 = a1.toString();
String b2 = a2.toString();
String b3 = a3.toString();
String b4 = a4.toString();
String b5 = a5.toString();
String b6 = a6.toString();
Character c = 'a';
String b7 = c.toString();
String b8 = a8.toString();
System.out.println(b1+"...."+b2+"...."+b3+
"...."+b4+"...."+b5+"...."+b6+"...."+b7+"...."+b8);
System.out.println(b1+b2);
//string data to primitive datatype
byte c1 = Byte.parseByte("100");
short c2 = Short.parseShort("234");
int c3 = Integer.parseInt("122");
long c4 = Long.parseLong("222");
float c5 = Float.parseFloat("6666.88f");
double c6 = Double.parseDouble("23.34");
//char c7 = Character.parseChar("a");
boolean c8 = Boolean.parseBoolean("true");
boolean c9 = Boolean.parseBoolean("true1");
System.out.println(c1+"...."+c2+"...."+c3+
"...."+c4+"...."+c5+"...."+c6+"...."+c8+"...."+c9);
```

```
//pdt to string

String d1 = String.valueOf(127);

String d2 = String.valueOf(127);

String d3 = String.valueOf(127);

String d4 = String.valueOf(127l);

String d5 = String.valueOf(127.89f);

String d6 = String.valueOf(127.78d);

String d7 = String.valueOf('a');

String d8 = String.valueOf(false);

System.out.println(d1+"...."+d2+"...."+d3+

"...."+d4+"...."+d5+"...."+d6+"...."+d7+"...."+d8);

System.out.println(d1+d8);

//System.out.println(127+false);

String e1 = Byte.toString((byte)100);

String e2 = Short.toString((short)200);

String e3 = Integer.toString(100);

String e4 = Long.toString(100l);

String e5 = Float.toString(223.34f);

String e6 = Double.toString(34.45d);

String e7 = Character.toString('a');

String e8 = Boolean.toString(false);

System.out.println(e1+"...."+e2+"...."+e3+
```

```
    "...."+e4+"...."+e5+"...."+e6+"...."+e7+"...."+e8);  
    System.out.println(e5+e6);  
}  
}
```

Auto Boxing: converting or representing primitive data to object data directly is called auto boxing.

```
ex: Integer i= 10;  
    Boolean b = false;
```

Auto Unboxing: converting or representing wrapper class to primitive data type data directly is called auto unboxing.

```
ex:  
int i = new Integer(100);  
boolean b = new Boolean(100);
```

Above statements valid from java 1.5 version.

Packages:

Package is a collection/group of classes, interfaces, enums, and sub-packages.

Some of the packages are given by java software and some of the packages are given by compiler.

The packages which are given by java s/w are called predefined packages.

The packages which are given by compiler or created by the compiler are called user defined packages.

Package is a special java folder.

By using “package” keyword we can develop our own packages.

Syntax:

```
package packagename;
```

Example:

```
package p1;
```

Package is a second section of our program, first section is comment section.

Package statement is always first statement in java program.

Package statement is an optional statement but in the real time every “.java” file having package keyword.

The compilation and execution of the package program is different from normal program.

Compilation phase:

If we want to compile the java package related program we must use below option.

Syntax: javac -d . FileName.java

- Here “-d” option will specify the folder creation. We are giving information to compiler to create one folder with the package name called nit.

- Here dot (“.”) operation will specify, current directory that means in nit folder and place the “dot class” file in that nit package.

Execution:

We should use the package name before the class name.

Syntax: java packagename.classname

Creating package in current working directory:

```
package nit;
```

```
class A{  
    public static void main(String...args){  
        System.out.println("this is package program");  
    }  
}
```

Filename: A.java

compilation: javac -d . A.java

execution: java nit.A

Creating package in some other directory:

```
package google;  
  
class B{  
    public static void main(String...args){  
        System.out.println("this is package program");  
    }  
}
```

FileName: B.java

Compilation: javac -d otherdirectorypath filename.java

Ex: javac -d E:\cj B.java

Execution: other directory packages can be execute in two ways

- 1) shift out from current working directory to package existing directory.

```
C:\Users\lenovo\Desktop\jse(7to9)> cd E:\cj(enter)
```

```
E:\cj> java nit.B
```

2) setting the classpath

```
C:\Users\lenovo\Desktop\jse(7to9)>
set classpath=.;E:\cj;
java nit.B
```

Creating sub packages:

```
package nit.cj.ram;
class C{
public static void main(String...args){
System.out.println("sub packages");
}
}
```

Filename: C.java

Compilation: javac -d . C.java

Execution: java nit.cj.ram.C

Access Modifiers:

Accessibility modifiers will provide permission to developer to use the variable, methods, class, interface, enum, abstract class.

In java, we have four types of accessibility modifiers, they are

- private.
- default. (package-private)

- protected.
- public

private:

private data can be access only with in the class level. We cannot acces out side of the class with in the same .java file, with in the same package or within the outside package.

default:

If we are not declared any access modifier in front of the variable, that is comes under default data.

From java 1.8 onwards we can default keyword in front of the methods of interface.

default data can be accessible with in the same class, with in the same .java file, with in the same package, but not accessible within the outside package class.

protected: protected can be access within the same class, with in the same .java file, with in the same package, but not accessible within the outside package class.

protected data some time acting as default data and sometimes as an public.

If other package class subclass of this class and data is static type, then protected data can be access outside of the package also.

Note:

Packages are provides security to both default and protected data. Packages provide 100% security to default data and protected data. But some time packages are not provide security to protected data.

public: public data can be access anywhere in the project.

```
package ram;

public class A {

    private int a = 111;

    int b = 222;

    protected int c = 333;

    protected static int d = 444;

    public int e = 555;

    public static void main(String[] args) {

        A obj = new A();

        System.out.println(obj.a);

        System.out.println(obj.b);

        System.out.println(obj.c);

        System.out.println(obj.e);

    }

}
```

Filename: A.java

Compilation: javac -d . A.java

Execution: java ram.A

```
package ram;

public class B {

    public static void main(String[] args) {

        A obj = new A();

    }

}
```

```
//System.out.println(obj.a);
System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj.e);

}

}
```

Filename: B.java

Compilation: javac -d . B.java

Execution: java ram.B

```
package sam;

import ram.A;

public class C extends A{
    public static void main(String[] args) {
        A obj = new A();
        //System.out.println(obj.a);
        //System.out.println(obj.b);
        //System.out.println(obj.c);
        System.out.println(obj.d);
        System.out.println(obj.e);

    }
}
```

Filename: C.java

Compilation: javac -d . C.java

Execution: java sam.C

```
Bank.java          Desktop
package com;
public class Bank{
    public static String bankName="sbi";
    static{
        System.out.println("Bank static block");
    }
}

Employee.java
package nit;
import com.Bank;
class Employee{
    public static void main(String[] r){
        System.out.println("RAMCHANDRA");
        System.out.println(Bank.bankName);
    }
}
```

```
javac -d D:\html Bank.java
set classpath=.;D:\html;
javac -d E:\CoreJavaMorning Employee.java
set classpath=.;D:\html;E:\CoreJavaMorning;
java nit.Employee
```

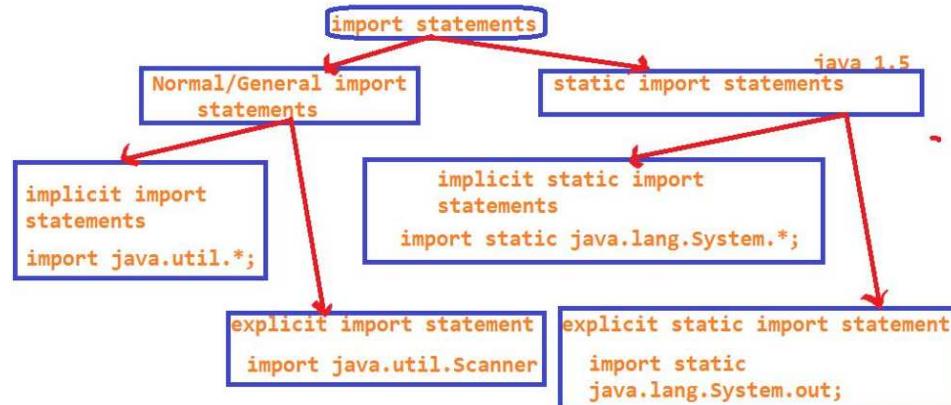
import statements are two types.

1. static import statements.
2. non-static/General import statement

General import statement can loads only classes, interface, abstract class, annotations, enum but not its members.(fields, methods).

Static import statements import the static fields, static methods. These variables and members must be public.

It is introduced in java 1.5 version.



Note:

Fully Qualified Name=
packagename+classname/interfacename/enumname/abstractclass/annotationname;

```
package sam;

/*import static java.lang.System.out;
import static java.lang.System.err;
import static java.lang.System.currentTimeMillis;
import static java.lang.System.identityHashCode;
import static java.lang.System.exit;*/
import static java.lang.System.*;

class B{

}

public class C{
    public static void main(String[] args) {
        System.out.println("static out filed");
        out.println("with out system class");
        err.println("err field");
        System.out.println(System.currentTimeMillis());
        out.println(currentTimeMillis());
        B obj = new B();
        out.println(System.identityHashCode(obj));
        out.println(identityHashCode(obj));
    }
}
```

```
    exit(0);

    System.out.println("program terminated in the previous statement");

}

}
```

Difference between #include and import statement?

#include will copy the code from library to program. so internally the code size will be increased.

import statement will give only response to java program.

Difference between import java.util.ArrayList and java.util.*;

In the first import statement we can access only ArrayList and its super class and interface functionalities.

In the second import statement we can access all classes and interface and abstract class related to util package.

Java archive: (jar)

It is one compressed file in java standard edition.

It contains java standard edition classes.

It can reduce file size.

With the help jar we can execute the programs also.

Web archive: (war).

It is also one compressed file in advanced java.

It contains the following files.

.servlet related classes

.jsp file

.html files

.jpeg files

.xml files

.properties files

Enterprise archive: (ear)

It is also one compressed file in enterprise edition.

This file contains the following files.

.servlet related classes

.jsp

.html

.jpeg

.xml files

.properties files and enterprise java beans classes.

Creating .jar file and add .class file:

```
public class A{  
    public static void main(String... args){  
        System.out.println("jar file");  
    }  
}
```

```
javac A.java
```

We are getting A.class file

```
jar -cf one.jar A.class
```

Delete the A.class file

Set the one.jar file in the class path then execute the program

java A

output: jar file.

```
public class B{
```

```
    public static void main(String... args){
```

```
        System.out.println("updating jar file");
```

```
}
```

```
}
```

javac B.java

We are getting B. class file

add B.class file to one.jar

jar -uf one.jar B.class

jar -tf one.jar

META-INF/MANIFEST.MF

A.class

B.class

java A

java B

Executing the java program with help of javaw command.

```
java.io.*;  
  
class D{  
  
public static void main(String...ram)throws IOException{  
  
FileOutputStream fos = new FileOutputStream("ram1.text");  
  
PrintStream ps = new PrintStream(fos);  
  
ps.write('d');  
  
ps.write('e');  
  
}  
  
}
```

```
javac D.java
```

```
javaw D
```

creating executable jar.

- 1) Write one package program.

```
package suji;  
  
public class A{  
  
public static void main(String...ram){  
  
System.out.println("executablejar");  
  
}  
  
}  
  
filename A.java
```

2) Compile our package program with below syntax.

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

```
javac -d . A.java
```

3) Take one notepad write below code in that.

Main-Class: suji.A

4) Save above notepad with MANIFEST.MF

5) Add MANIFEST.MF file with the help of bellow code.

```
jar -cvfm ud.jar MANIFEST.MF suji
```

6) Type bellow in the command prompt

```
java -jar ud.jar.
```

Note: class (A) must be public

In the MANIFEST.MF file M,C must be capital letter after ':' we need give one space and after packagename.classname we need to click on enter button.

ud.jar name is userdefine.

Executing the java program by using batch file.

1) write one program

```
package rabi;  
public class A{  
    public static void main(String...ram){  
        System.out.println("batchfile");  
    }  
}
```

filename A.java

2) take one notepad

write bellow code in that.

```
javac -d . A.java
```

```
java rabi.A
```

```
pause
```

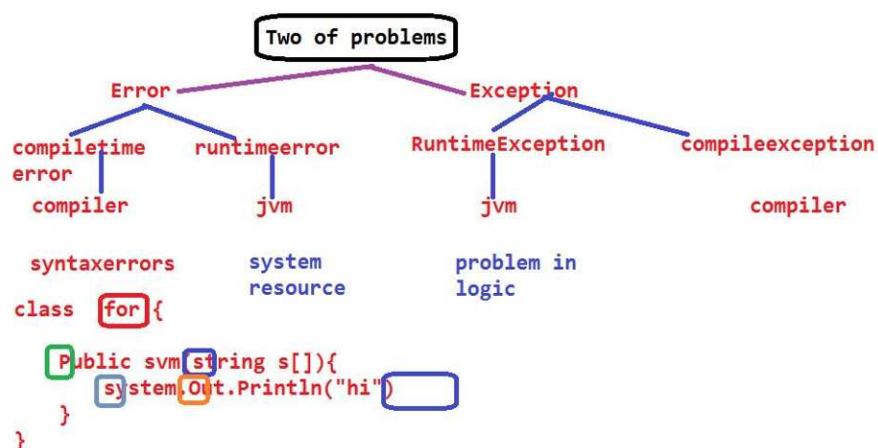
And save with filename.bat

ex: run.bat

3) double click on run.bat

Automatically command prompt will be open and compile the program and executing the program and display the output.

Exception Handling:



Exception: Exception is an event. It can be raised by the JVM at runtime due to problem of logic.

Whenever exception raises in the program, then program execution stopped and remaining statements are not executed.
Some times we need to execute some necessarily executable statmenets like dbconnection, file connection, etc...

Whatever the exception message given by java those are not understandable by the enduser, so we need provide so user understandable exception messages then we go for EXCEPTION HANDLING.

Some of logical problems are:

```
class MM{  
    public static void main(String[] s){  
        String s1 = s[0];  
        String s2 = s[1];  
        System.out.println(s1+s2);  
    }  
}
```

javac MM.java

valid execution statements:

```
java MM 123 234  
java MM ram sam  
java MM "%^&*(" "@#$%^"
```

Exception raised execution statements:

```
java MM (click on enter)  
java.lang.ArrayIndexOutOfBoundsException--0
```

```
java MM 123 (enter)
java.lang.ArrayIndexOutOfBoundsException--1

class MM{
    public static void main(String[] s){
        String s1 = s[0];
        String s2 = s[1];           System.out.println(s1+s2);
        int i = Integer.parseInt(s1);
        int j = Integer.parseInt(s2);
        System.out.println(i+j);
    }
}

javac MM.java
java MM 123 234
o/P: 123234
357

java MM ram sam
o/p: ramsan

java.lang.NumberFormatException
```

```
class MM{
    int a =111;
    void m1();
    public static void main(String... ram){
        System.out.println("Main Method");
    }
}
```

```
    MM obj = new MM();
    obj = null;
    //System.out.println(obj.a);
    obj.m1();
}
}
```

Note:

On top of null reference, we cannot call any variables, methods.

If we call then we will get java.lang.NullPointerException.

```
class MM{
    public static void main(String... ram){
        System.out.println("Main Method");
        //Object o = new String(); //upcasting
        Object o = new Object();
        String s1 = (String)o; //downcasting
    }
}

java MM  java.lang.ClassCastException

class MM{
    public static void main(String... ram){
        System.out.println("Main Method");
        int a [] = new int[-6];
    }
}
```

```
}
```

Array dimensions must be positive or zero not negative.

If we mention size in negative mode then we will get
java.lang.NegativeArraySizeException.

Before doing the down casting, first we should do up casting otherwise we will get java.lang.ClassCastException

```
class MM extends Thread{  
  
    public void run(){  
        System.out.println("this is run method");  
    }  
  
    public static void main(String[] s){  
        MM obj = new MM();  
        obj.start();  
        obj.start();  
    }  
}  
  
javac MM.java  
  
java MM  
this is run method  
java.lang.IllegalThreadStateException  
class MM {  
    public static void main(String[] s){
```

```
int a = 10/0;
}

}

javac MM.java

java MM

java.lang.ArithmeticException

class MM {

    int a =111;

    public static void main(String[] s){

        MM obj = new MM();

        obj=null;

        System.out.println(obj.a);

    }

}

javac MM.java

java MM

java.lang.NullPointerException

class MM extends Thread{

    public void run(){

        System.out.println("this is run method");

    }

    public static void main(String[] s){
```

```
    MM obj = new MM();
    obj.setPriority(11);//legal priorities from 1 to 10
    obj.start();
}

javac MM.java
java MM
java.lang.IllegalArgumentException
class MM {
    public static void main(String[] s1){
        Object o = new Object();
        String s = (String)o;
    }
}
javac MM.java
java MM
class MM{
    public static void main(String... ram){
        System.out.println("Main Method");
        String s = "internationalization";//i18n
        System.out.println(s.charAt(20));
    }
}
```

In above program, we dont have 20th index position in string s.

But we tried to print the values that why we are getting

java.lang.StringIndexOutOfBoundsException

java.lang.ClassCastException

float a = 10/0.0f-->infinity

float b = -10/0.0f--> -infinity

float c = 0/0.0f--->NaN

float d = 0./0; -->NaN

Error: Error is an event which can be raised by jvm at runtime due to problem of lack of system resource.

```
class MM {  
    MM obj = new MM();  
    public static void main(String[] s1){  
        MM obj = new MM();  
  
    }  
}
```

javac MM.java

java MM

java.lang.StackOverflowError

Note:

JVM version always greater than or equal to compiler version, otherwise we will get one runtime error that is:

UnsupportedClassVersionError

```
class MM {  
    static int a = 10/0;  
    public static void main(String[] s1){  
    }  
}  
  
javac MM.java  
  
java MM  
  
java.lang.ExceptionInInitializerError
```

Exception is a event, which can be raised by jvm at runtime due to the problem of logic.
some of examples for exceptions:

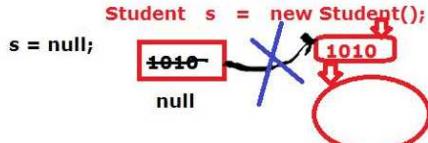
```
int a = 10/0; //Infinity java.lang.ArithmetricException  
-2147483648 to 0 to 2147483647  
  
String s = "vennela";  
sop(s.charAt(7)); //java.lang.StringIndexOutOfBoundsException  
  
int[] a = new int[-1]; //java.lang.NegativeArraySizeException  
  
int[] b = {11,22,33,44,55};  
s.o.p(b[5]); //java.lang.ArrayIndexOutOfBoundsException  
  
Object o = new Object();  
  
String s1 = (String)o;//downcasting //java.lang.ClassCastException
```

What is the null?

--> Literal

What is the use of null?

--> To make our referenced/used memory into unreferenced/unused for memory cleaning by the GC.



What is the type of null?

--> The null type is any referenced type.

```
int a[] = null;  
Student s = null;  
interface i = null;  
enum e = null;
```

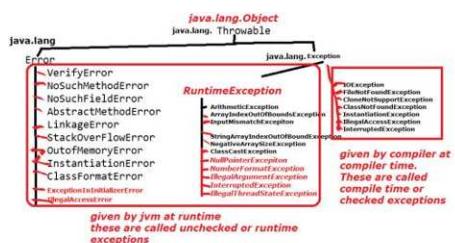
What type of operations are can we do on null?

- 1) Assign the value to reference
 - 2) Type casting

compiler always concentrate on variable or value type whereas jvm concentrate on value or memory.

```
class A{  
    int a = 111;  
    void m1(){  
        public static void main(String[] s){  
            System.out.println(10);  
            System.out.println(false);  
            //System.out.println(null);  
            System.out.println((String)null); 2.Type Casting  
            A obj = null; 1.Assign the value to reference variable  
            System.out.println(obj);  
            //System.out.println(obj.a); NPE  
            //obj.m1(); NPE  
            System.out.println(++null); CE:UnExpected Type  
        }  
    }  
}  
null can be assign to only referenced variable not for  
primitive variable.  
int x = null; //CE: incompatible types
```

Exception Hierarchy:



Use of Exception Handling:

- 1) Whenever exception is raised internally JVM provides some predefined exception message.

Those messages are not understandable by the end user.

If we want to provide user friendly message then we can go for exception handling.

- 2) Whenever exception is raised automatically the remaining statements are not executed.

Program abnormally terminated (without giving information to programmer).

So some necessary executable statements are not executed.

If we want to execute all necessary statements then we can go for exception handling.

We can achieve exception handling mechanism with the help of following keywords

those are

1. try
2. catch
3. finally
4. throw
5. throws
6. assert

try : In this block we should always write exception raised statements.

If we don't know which statements raise exception then write all the statements in try block.

catch: In this block we should hold exception object which was raised in the try block.

This block execution is depends upon the try block.

If try block not raised exception then catch block will not be executed.

If try block raised exception then catch block will be executed.

Note: catch block always holds exception objects not normal objects.

finally: This block is not depends upon try block result.

If try block is raised the exception or not, the finally block always executed.

This block is used write the statements which are necessary to execute.

Like database connection, file closing.....

This block is always executed.

When finally block not executed?

If programmer write System. exit (0), that means if we are explicitly stop the program then finally block not executed.

If JVM execution suddenly stopped then finally block not executed.

Valid combinations between try and catch and finally blocks:

(1)

```
try{
```

```
}
```

```
catch(-){
```

```
}
```

```
finally{
```

```
}
```

(2)

```
try{
```

```
}
```

```
finally{
```

```
}
```

(3)

```
try{
```

```
}
```

```
catch(-){
```

```
}
```

(4)

```
try{
```

```
}
```

```
catch(){
```

```
}
```

```
catch(){
```

```
}
```

```
catch(){
```

```
}
```

```
finally{
```

```
}
```

```
-----
```

```
(5)
```

```
try{
```

```
}
```

```
catch(){
```

```
}
```

```
catch(){
```

```
}
```

```
catch(){
```

```
}
```

```
-----
```

```
(6)
```

```
try{
```

```
    try{
```

```
}
```

```
    catch(-){
```

```
}
```

```
    finally{
```

```
}
```

```
}
```

```
    catch(-){
```

```
        try{
```

```
}
```

```

catch(-){

}

finally{

}

}

finally{

try{

}

catch(-){

}

finally{

}

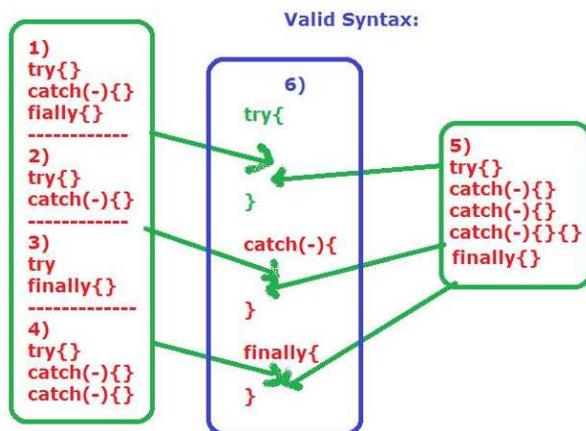
}

```

```

1.try
2.catch
3.finally
=====
4.throw
5.throws
6.assert(java 1.4)

```



Invalid statements:

```

try{

}

finally{

```

```
}

catch{

}

*****



try{

}

*****



catch(){

}

*****



finally{



}

*****
```

```
try{



}

finally{



}

finally{



}

*****



try{



}System.out.println("we cannot write");

catch(-){
```

```

}System.out.println("we cannot write");

finally{

}

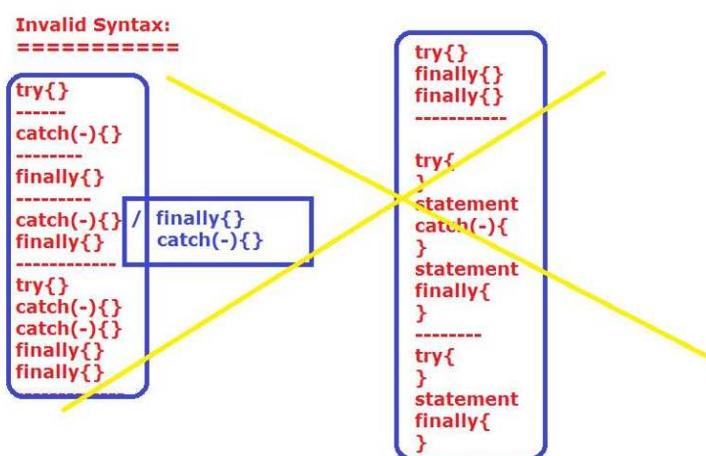
*****
try{

}

System.out.println("we can not write");

finally{
}

```



Internal flow of Exception:

Whenever exception is raised in the program JVM will do the following things.

1. JVM will check behavior/characteristic of an exception.

```
int a = 10/0;
```

2. Based on characteristic of an exception, JVM will select one appropriate exception class.

```
java.lang.ArithmaticException
```

3. After selecting the class, JVM will create an object for that selected class.

```
ArithmaticException ae= new ArithmaticException();
```

4. While creating an object for that JVM will add description of an exception.

like "/ by zero".

5. Finally JVM will handover to catch block or print on the console.

Exception in thread "main" java.lang.ArithmaticException: / by zero

at collection.MapDemo.main(MapDemo.java:8)

-->catch block parameter type must be required three types of functionalities, those are

1.Object functionalities

2.Throwable functionalities

3.Exception functionalities

We have three ways(Methods) to print exception messages.

1. printStackTrace().

2. toString()

3. getMessage().

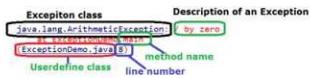
1.printStackTrace():

It will print exception message in the following format

```
java.lang.ArithmaticException: / by zero
```

at exception.ExceptionDemo.main(ExceptionDemo.java:8)

It will print predefine exception class fully qualified name, description of an exception fully qualified user define class name, method name, .java file name, line number, package name.



2. `toString()`:

java.lang.ArithmaticException: / by zero

It will print predefine exception class fully qualified name, description of an exception

3. `getMessage()`:

/ by zero

It will print only description of exception

```
public class ExceptionDemo {  
    public static void main(String[] args) {  
        //int a = 10/0;  
        //System.out.println("these statements are not executed");  
        try{
```

```
        int a = 10/0;

    }

    catch(Exception e){

        e.printStackTrace();

        System.out.println(e.toString());

        System.out.println(e.getMessage());

    }

    System.out.println("These statements are executed");

}

}
```

```
public class ExceptionDemo {

    public static void main(String[] args) {

        try{

            int a = 10/0;

        }

        catch(Exception e){

            System.out.println("DONT ENTER ZERO AS AN
DENOMINATOR");

        }

    }

}
```

In the first program we are execute the all the statements
in this program we did give user understandable exception message.

```
public class ExceptionDemo {  
    public static void main(String[] args) {  
        try{  
            int a = 10/0;  
            System.out.println("not executed");  
        }  
        //System.oyut.println("we cannot write the statement  
        //      b/w try and catch")  
        catch(Exception e){  
            System.out.println("DONT ENTER ZERO AS AN DENOMINATOR");  
            //System.exit(0);  
            //in the above line we did explacitly stop the program  
        }  
        //System.oyut.println("we cannot write the  
        //statement b/w finally and catch")  
        finally{  
            System.out.println("this block is always executed");  
        }  
        System.out.println("we can write the statements " +  
            "after the finally block");  
    }  
}  
public class ExceptionDemo {
```

```
public static void main(String[] args) {  
    try{  
        //int a = 10/0;  
        //Object o = new Object();  
        //String s = (String)o;  
        //int a [] = {10,20,30};  
        //System.out.println(a[3]);  
        //int a[] = new int[-1];  
        Object[] s = new Integer[4];  
        s[0] = 4.4;  
    }  
    //catch(Exception e){}  
    catch(ArithmeticException e){  
        System.out.println("dont enter zero as an denominator");  
    }  
    catch(ClassCastException e){  
        System.out.println("Super class memory we can not" +  
            " place into subclass reference ");  
    }  
    catch(ArrayIndexOutOfBoundsException e){  
        System.out.println("we dont have any element in 3rd " +  
            "index position");  
    }  
    catch(NegativeArraySizeException ne){
```

```

        System.out.println("we can not create array" +
                           " with negative values");

    }

    catch(Exception e){

        System.out.println("we can not place double value in " +
                           "integer array");

    }

    finally{

        System.out.println("this block is always executed");

    }

}

```

If we are writing more than one catch blocks, java.lang.Exception class type parameter must be placed in last catch block.

The reason is whatever the exception is raised in the catch block that is always matched with Exception class type. (upcasting).

So the remaining catch blocks will not be executed.

```

public class ExceptionDemo2 {

    public static void main(String[] args) {

        try{
            System.out.println("outer try block");

            try{
                System.out.println("inner try block");

                int a= 10/0;

            }
        }
    }
}

```

```

/*catch(Exception e){

    System.out.println("inner catch block");

    System.out.println("dont enter zero as deono");

}*/



finally{

    System.out.println("inner finally block");

}

}

}

catch(Exception e){

    System.out.println("outer catch block");

}

finally{

    System.out.println("outer finally block");

}

}

}

}

```

Note: If any exception is available in inner try block first jvm check inner catch block.

If available then inner catch will be execute. If not available or not matched then JVM checks outer catch block.

If available then outer catch block will be executed, if not available or parameter type exception class is not supports raised exception class (not matched) then raised exception class object handover to JVM.

```

public class ExceptionDemo2 {

    public static void main(String[] args) {

```

```
try{  
    System.out.println("outer try block");  
    try{  
        System.out.println("inner try block");  
        int a= 10/0;  
    }  
    /*catch(Exception e){  
        System.out.println("inner catch block");  
        System.out.println("dont enter zero as deono");  
    }*/  
    finally{  
        System.out.println("inner finally block");  
    }  
}catch(Exception e){  
    System.out.println("outer catch block");  
}  
finally{  
    System.out.println("outer finally block");  
}  
}
```

Note:

finally block not only override the return value but also override the exception.

```
package exception;

class Student{

int sid = 111;

String sname = "ram";

void m1()throws Throwable{

    Student s1 = this;

    System.out.println(s1.sid);

    System.out.println(s1.sname);

    s1 = null;

    try{

        System.out.println(s1.sid);//NullPointerException

        }catch(Exception e){

            //throw e;

            throw e.initCause

            (new ClassCastException("we are " +      "calling variable on null

reference"));

        }

    }

}
```

```
public class ChinedException {

    public static void main(String[] args) {

        Student s = new Student();
```

```
try {  
    s.m1();  
} catch (Throwable e) {  
    System.out.println(e.getCause());  
}  
}  
}
```

Java provides mainly two types of functionalities those are
Normal Functionalities
Exception functionalities

Whatever functionalities are given by java some time not support for our project requirement, then we should go for develop our own functionalities.

To develop our own exception functionalities, java provides following flexibilities.

How to develop user define exception:

1. Extracting functionalities from `java.lang.Exception` class.
2. Define zero parameterized constructor
3. Define parameterized constructor
4. Checking the behaviour/condition/characteristic
5. Select one user define class
6. Create object for selecting class
7. Meanwhile of object creation, add the reason of the exception
8. Handover to either catch block or console.

```
package exception;  
  
class MyException extends Exception{  
  
    MyException(String msg){
```

```
super(msg);

}

MyException(){

}

void getAge(int age)throws MyException{

try{

    if(age >= 18){

        System.out.println("This person is elegible for voting");

    }

    else

        throw new MyException("This person is not elegible for voting");

}catch(Exception e){

    throw e;

}

}

}

public class UserDefineExceptionDemo {

    public static void main(String[] args)throws MyException {

        MyException me = new MyException();

        me.getAge(18);

        /*try{

            me.getAge(17);

        }catch(Exception e){

        }*/

    }

}
```

```
 }  
}
```

If we want to develop user define exceptions, programmer should do the following things.

- 1) Programmer should take one user define class.

Ex: class MyException{

```
}
```

- 2) Creating an object for that class.

```
new MyException("This person is not eligible for voting");
```

- 3) If we want to add exception message to object we need some logic, that logic is available in java.lang.Exception .

- 4) If we want to forward our control and exception message from sub class to super class we need bellow syntax.

```
ex: MyException(String msg){  
    super(msg);  
}
```

- 5) If exception is predefine, then JVM will handover that exception object from try block to catch block.

Here exception is user define, so we should give the information to JVM about our exception object.

If we want handover the object from try block to catch block we need "throw" keyword.

With the help of throw keyword we can give information to JVM about our object that is make our object as an exception object.

And also with the help of throw keyword we can send exception object from called method to calling method.

If any method having throw keyword in its catch block that method must be use throws keyword.

throw: "throw" keyword is used to forward the exception object from one method to another method within the application explicitly.

throws: If any compile time exception raised in our program, then we need to handle those exceptions. If we are not interest to handle, then we need handover those exceptions to JVM. "throws" is used to forward the exception object from java application to JVM application then we can go for "throws" keyword.

With the help of throws developer provide some information to compiler that is there may be chance of raising exception in our method.

Again with the help of throws keyword compiler will give information to programmer as a compiler time exception.

If are using any method, which is raised exception, in that time compiler will throw compile time/checked exception.

throw:

throw is used forward the exception object from try block to catch block as well as one method to another method.

throw is also used for giving information to jvm about our(user define) exception object.

throws:

Programmer will give information to compiler about there may be a chance of raising the exception in the method (or) By seeing the throws keyword by compiler, it came one decision there may be chance of raising the exception, that information is giving by the compiler in the form checked exception.

throws is also handover exception object from our application to jvm.

We can handle checked exception in the following two ways.

1. try and catch block
2. throws keyword.

```
public class ExceptionDemo {  
    static int m1(){  
        try{  
            System.out.println("try block");  
            //int a = 10/0;  
            return 111;  
        }  
        catch(Exception e){  
            System.out.println("catch block");  
            //return 222;  
        }  
        finally{  
            System.out.println("finally block");  
            //return 333;  
        }  
        System.out.println("unable to write");  
        return 444;  
    }  
    public static void main(String[] args) {  
        System.out.println(m1());  
    }  
}
```

Note:

If try and catch and finally blocks having return statement, we can't write any statements after the finally block.

try and catch block return statement values always replaced with finally block return statement value.

If try and catch block not having any return statement, after the finally block we can write some other statements (optional) and also we should write return statement (mandatory) also.

Additional points:

If any method having return statement:

1. We should write return statement in try and catch block.
2. If try and catch block having return statement, we can't return statement after the catch block.
3. If try only having only return statement we need to return statement after the catch block.
4. If catch block having only return statement, we need to return statement after the catch block.
5. try and catch block return statements values always override by the finally block return statement value.

```
package nit.cj.sb;

import java.util.Scanner;

class InvalidAgeException extends java.lang.Exception{
    InvalidAgeException(String message){
        super(message);
    }
    InvalidAgeException(){}
    public void getAge(int age) throws Exception {
        try{
            if(age >= 18){
                System.out.println("you are elegible for voting");
            }
            else {

```

```

        throw new InvalidAgeException("invalid age");
    }
}
catch(Exception e){
    //e.printStackTrace();
    throw e;
}
}
}

public class ExceptionDemo {
    public static void main(String[] args)
    {//throws Exception{
        InvalidAgeException iae = new InvalidAgeException();
        Scanner scan = new Scanner(System.in);
        System.out.println("enter your age");
        byte age = scan.nextByte();
        try{
            iae.getAge(age);
        }catch(Exception e){
            //e.printStackTrace();
            System.out.println("Sorry! You Dont have age greater than
17");
        }
    }
}

```

Why can't we write Object type parameter in catch block?

- A) Catch block always executing meanwhile of exception raises in the try block.
- B) Try block always throwing execption objects, which are directly or indirectly subclass of Throwable class.
- C) So catch block parameter type required both Object class functionalaites and Throwable class functionalities.
- D) If we are writing

```

    catch(Object o){
    }

```

Parameter 'o' only hava Object class functionalities but doesnot Throwabe functionalities.

E) Thats why we can not write Object type parameter in catch block.

In which scenario finally block not executed?

A) If programmer writing System.exit(0) then automatically jvm functionalities will stop, so finally block not executed

```
public class Demo {  
  
    public static void main(String[] args) throws Exception{  
        try{  
            System.out.println("try block");  
            int a = 10/0;  
            System.exit(0);  
        }  
        catch(Exception e){  
            System.out.println("catch block");  
            System.exit(0);  
        }  
        finally{  
            System.out.println("finally block");  
        }  
    }  
}
```

B) Jvm suddenly crash.

Exception Chaining:

```
public class ExceptionDemo {  
  
    int a = 111;  
  
    static void m1()throws Throwable{  
  
        try{  
  
            ExceptionDemo ed = new ExceptionDemo();  
  
            System.out.println(ed.a);  
  
            ed = null;  
        }  
    }  
}
```

```

        System.out.println(ed.a);

    }

    catch(Exception e){
        throw e.initCause(new ArithmeticException("we are calling" +
            " variable a on top of null reference"));
    }

}

public static void main(String[] args) {
    int a = 10/0;
    try {
        m1();
    } catch (Throwable e) {
        //e.printStackTrace();
        System.out.println(e.getCause());
    }
}

```

One Exception class provide the information like cause of exception of another exception class is called exception chaining.

If we want to add exception message to existed object we have one method that is "initCause()".

If we want read the exception message, we have one more method that is "getCause()".

Exceptions can be classified as follows.

Fully checked exceptions:

Compiler will check class, and all its subclasses is called FCE. All compile time exceptions are comes under fully checked exceptions only.

Partially checked exceptions:

Compiler will check class and not check all its subclass is called PCE.

Ex: `java.lang.Exception,`

`java.lang.Throwable`

If method having return type and having try and catch blocks, we need to write return statement with value in both blocks.

If method having return type and having try and catch and finally blocks we no need to write return statement with values in try and catch blocks. Directly we can write return statement with value in finally block.

If try and catch having return statements then finally block return statement always overrides the try and catch block return statement.

If finally block having return statement we can't write any other statement after the finally block.

We can write some statements after finally block in the following conditions.

Only try block having return statement (or)

Only catch block having return statement.

If no block having return statement.

`package exception;`

```
public class ExceptionDemo {
```

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

    try{
        System.out.println("outer try block");

        try{
            System.out.println("inner try block");
            int a[] = {1,2,3};
            System.out.println(a[3]);
        }catch(ArithmeticException e){
            System.out.println("inner catch block");
        }
        finally{
            System.out.println("inner finally block");
        }
    }catch(NullPointerException e){
        System.out.println("outer catch block");
    }
    finally{
        System.out.println("outer finally block");
    }

    System.out.println("*****");
    try{
        System.out.println("outer try block");

        try{

```

```
        System.out.println("inner try block");

        System.out.println("ram".charAt(3));

    }

    finally{

        System.out.println("inner finally block");

    }

}catch(Exception e){

    System.out.println("outer catch block");

}

finally{

    System.out.println("outer finally block");

}

System.out.println("*****");

try{

    System.out.println("outer try block");

    try{

        System.out.println("inner try block");

        int a[] = new int[-1];

    }catch(Exception e){

        System.out.println("inner catch block");

    }

    finally{

        System.out.println("inner finally block");

    }

}
```

```
 }catch(Exception e){  
    System.out.println("outer catch block");  
}  
  
finally{  
    System.out.println("outer finally block");  
}  
  
System.out.println("*****");  
  
try{  
    System.out.println("outer try block");  
    int a = 10/0;  
    try{  
        System.out.println("inner try block");  
    }catch(Exception e){  
        System.out.println("inner catch block");  
    }  
    finally{  
        System.out.println("inner finally block");  
    }  
}catch(Exception e){  
    System.out.println("outer catch block");  
}  
  
finally{  
    System.out.println("outer finally block");  
}
```

```
System.out.println("*****");
try{
    System.out.println("outer try block");
    try{
        System.out.println("inner try block");
    }catch(Exception e){
        System.out.println("inner catch block");
    }
    finally{
        System.out.println("inner finally block");
    }
}catch(Exception e){
    System.out.println("outer catch block");
}
finally{
    System.out.println("outer finally block");
}

}

}

static void m2(){
try{
    System.out.println("try block");
}
}
```

```
        catch(Exception e){  
            System.out.println("catch block");  
        }  
        finally {  
            System.out.println("finally block");  
        }  
        System.out.println("we can write");//valid  
    }  
  
*****
```

s.o.p statements are used to do the debugging.

if we are using sop for debug, before giving the project to the client
we need to delete sop's.

when ever we using sop the controle is goning console come back to
java app, it may be leads to time consuming.

with the help of sop there may be chance to guess our code by the
client.

to avoiding above drawback java introduced topic like assert in
the version java 1.4.

if assert statement give true then our controle is keep on going
otherwise we will get assertio n error.

IllegalAccessException:

Without changing `setAccessible(false)` to `setAccessible(true)` if we are trying to read private from outside of the class by using reflection api we will get `IllegalAccessException`

InstantiationException:

Loading the .class file dynamically and trying to create object for that class by using `newInstance()`, then that class must be have zero argument constructor.

```
public class A{
    public static void main(String[] r)
        throws Exception{
        Class cls = Class.forName(r[0]);
        Object o = cls.newInstance();
        B obj = (B)o;
        System.out.println(obj.a);
    }
}
```

```
class B{
    int a = 1111;
    B(int x){}
}
```

```
javac B.java
javac A.java
java A
java.lang.InstantiationException
```

`new` operator vs `newInstance()`:

`new` operator is used to create an object meanwhile developing program for a particular that means in the static manner.

`Student s = new Student();`

`newInstance()` is used to create an object dynamically.

That means meanwhile of developing program if we dont know about particular class name then we should go for `newInstance()`.

```
class Demo{
    public static void main(String[] r) throws Exception{
        //A obj = new A();
        Class cls = Class.forName(r[0]); Object o = cls.newInstance();
        if(o instanceof A){A obj = (A)o; System.out.println(" A class");
            System.out.println(obj.a); System.out.println(obj.b);
        }else{B obj = (B)o; System.out.println(" B class");
            System.out.println(obj.c); System.out.println(obj.d);
        }
    }
}
```

NoClassDefFoundError vs ClassNotFoundException:

```
class Demo{  
    public static void main(String[] r) throws Exception{  
        Class cls = Class.forName(r[0]);  
        Object o = cls.newInstance();  
        if(o instanceof A){  
            A obj =(A)o;  
            System.out.println(obj.a);  
            System.out.println(obj.b);  
        }  
        /*A obj = new A();  
        System.out.println(obj.a);  
        System.out.println(obj.b);*/  
    }  
}
```

meanwhile program execution, if we are sending classname dynamically, then jvm is unable to load that .class file then we will get `java.lang.ClassNotFoundException`

Declare class name directly in the .java file and meanwhile of executing the program if .class file not existed then we will get `NoClassDefFoundError`

java.lang.Object:

It is the super class for all predefines and user defined classes in java either directly or indirectly.

```
class A{  
}  
}
```

If we write class in the above manner java internally converts into as a sub class of `java.lang.Object` class.

```
javac A.java  
  
javap A  
  
class A extends java.lang.Object{  
    A(){  
        super();  
    }  
}
```

`java.lang.Object` class having bellow methods.

- 1) `getClass()`

- 2) `toString()`
- 3) `equals()`
- 4) `hashCode()`
- 5) `clone()`
- 6) `notify()`
- 7) `notifyAll()`
- 8) `wait()`
- 9) `wait(-)`
- 10) `wait(-,-)`
- 11) `finalize()`

Among the 11 methods, we can override 5 methods only.

Remaining methods are not participated in the method override concept.

Below methods are participated in the method override concept.

- 1) `toString()`
- 2) `eauals()`
- 3) `hashCode()`
- 4) `clone()`
- 5) `finalize()`

toString():

```
public String toString(){  
    return getClass().getName() + "@"+Integer.toHexString(hashCode());  
}
```

In the above syntax : getClass() method will create java.lang.Class reference with specific class bytecode.

getName() is the non-static method in java.lang.Class, with help of this method we will get class name and package information of an loaded class.

i.e: System.out.println(new Student());

s.o.p method internally calls toString() method.

This method(getClass().getName()) will print the output like "packagename.Student".

"@" --> whatever the code which is available in double quotes as it is printed on the console.

hashCode() method always return one unique identification number of memory in integer(decimal number system) format.

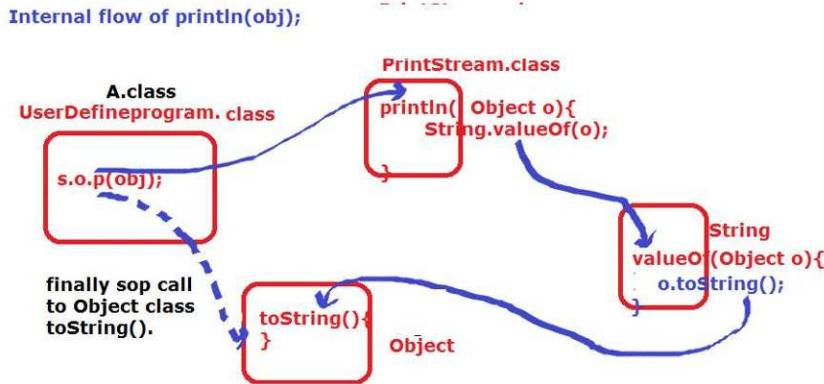
This integer number will converted into hexa decimal format with the help of toHexString() of java.lang.Integer class.

Ex: Assume if hashCode() return 159--> this 159 will converted into hexadecimal format like "9f".

finally we will get one output as

"packagename.Student@9f"

Note: Java avoids pointers concept with the help of toString().



```

package object;

class Employee{

    int eid;

    String ename;

    double esal;

    byte eage;

    Employee(int eid, String ename, double esal, byte eage){

        this.eid = eid;

        this.ename = ename;

        this.esal = esal;

        this.eage = eage;
    }

    public String toString(){

        System.out.println("control comes to toString()");

        //return getClass().getName()+"@"+Integer.toHexString(
        //hashCode());

        return eid+" "+ename+" "+esal+" "+eage;
    }
}
  
```

```
    }  
}  
  
public class ToStringDemo {  
    public static void main(String[] args) {  
        Employee e1 = new Employee(101,"ram",15000,(byte)28);  
        System.out.println(e1);  
    }  
}
```

hashCode():

This method provides a unique identification number of memory.

Whenever we use new keyword, JVM blindly provides new memory, that means new hashCode.

JVM is not check content to produce the hashCode.

JVM is always provides a new hashCode for different object.

If we want to provide hashCode based on content then we should go for override hashCode () .

"==" and equals () of Object class always calls JVM provides hashCode (). So if want call the user define hashCode () definitely we should override equals () of Object class.

equals (): This method is always checks hashCode() of objects, if hashCode of any two objects are different then equals() method returns false, otherwise returns true.

If we want to call our own hashCode () from equals (), we need to override the equals () in our class.

We can know the identification number of memory by using two ways.

- 1) hashCode ():

This method first preference gives to our own hashCode(), if not available Object class hashCode()(JVM provide memory)

2) identityHashCode();

This is static method of System class, it is always provides jvm provide hashCode(identification number of memory)

package object;

class Student{

 int sid;

 String sname;

 Student(int sid, String sname){

 this.sid = sid;

 this.sname = sname;

 }

 static int i=0;

 @Override

 public int hashCode(){

 i++;

 return sid+sname.hashCode();

 }

 public boolean equals(Object o){

 if(o instanceof Student){

 Student s = (Student)o;

 if(this.sid == s.sid && this.sname.equals(s.sname)){

 return true;

 }

```
        else return false;

    }

    else return false;

/*if(this.hashCode() == s.hashCode()){

    return true;

}

return false;*/

}

}

public class HashCodeDemo {

    public static void main(String[] args) {

        Student s1 = new Student(101,"ram");

        Student s2 = new Student(102,"sam");

        System.out.println(s1==s2);

        Student s3 = new Student(101,"ram");

        System.out.println(s1==s3);

        System.out.println("s1: "+s1.hashCode());

        System.out.println("s2: "+s2.hashCode());

        System.out.println("s3: "+s3.hashCode());

        System.out.println(Student.i);

        System.out.println("*****");

        System.out.println("jvm s1: "+System.identityHashCode(s1));

        System.out.println("jvm s2: "+System.identityHashCode(s2));

        System.out.println("jvm s3: "+System.identityHashCode(s3));
```

```
        System.out.println("*****");
        System.out.println(s1.equals(s2));
        System.out.println(s1.equals(s3));
    }
}
```

cloning:

Creating new object with updated data is called cloning.

In the cloning, we will get new object with new memory.

For doing cloning java provides one predefined method called `clone()`, this is protected method in `java.lang.Object` class.

`clone()` will throw one checked/compile time exception i.e
"`java.lang.CloneNotSupportedException`"

If any class data is participated in the cloning that class must be implements `java.lang.Cloneable` interface.

`java.lang.Cloneable` is a marker interface.

(An interface which doesn't contain any methods)

Whenever we do the cloning, non-static data are not going to be loaded.

(Creating new object with new memory with updated data without duplicate code is called cloning).

```
package object;

class Address implements Cloneable{
    String cityName = "hyderabad";
    String stateName = "telangana";
}

public class Student implements Cloneable{
```

```
int sid = 111;  
String sname = "ram";  
Address addr = new Address();  
public static void main(String[] args) throws  
CloneNotSupportedException{  
    Student s1 = new Student();  
    Student s2 = (Student)s1.clone();  
    System.out.println(s1.sid);  
    System.out.println(s1.sname);  
    System.out.println(s1.addr.cityName);  
    System.out.println(s1.addr.stateName);  
    System.out.println("*****");  
    System.out.println(s2.sid);  
    System.out.println(s2.sname);  
    System.out.println(s2.addr.cityName);  
    System.out.println(s2.addr.stateName);  
    System.out.println("*****");  
    s1.sid = 222;  
    s1.sname = "sam";  
    s1.addr.cityName = "vizag";  
    s1.addr.stateName = "AndhraPradesh";  
    System.out.println(s1.sid);  
    System.out.println(s1.sname);  
    System.out.println(s1.addr.cityName);
```

```
System.out.println(s1.addr.stateName);
System.out.println("*****");
System.out.println(s2.sid);
System.out.println(s2.sname);
System.out.println(s2.addr.cityName);
System.out.println(s2.addr.stateName);
System.out.println("*****");
s2.sid = 333;
s2.sname = "suji";
s2.addr.cityName = "sklm";
s2.addr.stateName = "AP";
System.out.println(s1.sid);
System.out.println(s1.sname);
System.out.println(s1.addr.cityName);
System.out.println(s1.addr.stateName);
System.out.println("*****");
System.out.println(s2.sid);
System.out.println(s2.sname);
System.out.println(s2.addr.cityName);
System.out.println(s2.addr.stateName);
System.out.println("*****");
}

}
```

Cloning is two types

1. Shallow cloning

2. Deep cloning.

Shallow cloning:

If we are doing updating on cloned object or existed object those changes will be effects to other objects is called shallow cloning.

Ex: Below program

```
public class A implements Cloneable{
    int a = 111;
    int b = 222;
    public static void main(String[] args)
    throws CloneNotSupportedException{
        A obj1 = new A();
        System.out.println("obj1: "+obj1.a+"...."+obj1.b);
        obj1.a = 333;
        obj1.b = 444;
        System.out.println("obj1: "+obj1.a+"...."+obj1.b);
        A obj3 = (A) obj1; //down casting
        System.out.println("obj3: "+obj3.a+"...."+obj3.b);
    }
}
```

```
package clonning;
class Address{
    String cityName = "Hyderabad";
    String stateName = "Telangana";
}
public class Student implements Cloneable{
    int sid = 101;
    String sname = "ram";
    Address addr = new Address();
    public static void main(String[] args)
    throws CloneNotSupportedException{
        Student s1 = new Student();

        System.out.println(s1.sid);
        System.out.println(s1.sname);
        System.out.println(s1.addr.cityName);
        System.out.println(s1.addr.stateName);

        s1.sid = 201;
        s1.sname = "yaane";
        s1.addr.cityName = "vizag";
        s1.addr.stateName = "AP";

        System.out.println("=====");
    }
}
```

```

        System.out.println(s1.sid);
        System.out.println(s1.sname);
        System.out.println(s1.addr.cityName);
        System.out.println(s1.addr.stateName);

        System.out.println("=====");
        Student s2 = new Student();

        System.out.println(s2.sid);
        System.out.println(s2.sname);
        System.out.println(s2.addr.cityName);
        System.out.println(s2.addr.stateName);

        System.out.println("=====");

        Object o = s1.clone();
        Student s3 = (Student)o;

        System.out.println(s3.sid);
        System.out.println(s3.sname);
        System.out.println(s3.addr.cityName);
        System.out.println(s3.addr.stateName);

        System.out.println("=====");
        s3.addr.cityName = "chennai";
        System.out.println(s3.addr.cityName);
        System.out.println("=====");
        System.out.println(s1.addr.cityName);

    }

}

```

class A{
 int a = 111;
 int b = 222;
 main(){
 A obj1 = new A();
 A obj2 = new A();
 obj1.a = 333;
 obj1.b = 444;
 }
}

in the above program whenever we create new object jvm will provide new memory with common data. But my requirement is we need to new memory with updated information of obj1 object, then we should go for CLONING

Deep cloning:

If we are doing updating on cloned object those changes will not be effected to existed object is called deep cloning.

Ex: Below program.

```
class Address implements Cloneable{
    String cityName = "Hyderabad";
    String stateName = "Telangana";
    public Address clone()throws CloneNotSupportedException{
        return (Address)super.clone();
    }
}
public class Student implements Cloneable{
    int sid = 101;
    String sname = "suji";
    Address addr = new Address();

    @Override
    public Student clone()throws CloneNotSupportedException{
        Student s3 = (Student)super.clone();
        this.addr = this.addr.clone();
        return s3;
    }

    public static void main(String[] args)
    throws CloneNotSupportedException{
        Student s1 = new Student();
        System.out.println("s1: "+s1.sid+".."+s1.sname+".."
                           +s1.addr.cityName+".."+s1.addr.stateName);
        System.out.println("====");

        s1.sid=201;
        s1.sname="ram";
        s1.addr.cityName="vizag";
        s1.addr.stateName="AP";
        System.out.println("s1: "+s1.sid+".."+s1.sname+".."
                           +s1.addr.cityName+".."+s1.addr.stateName);
        Student s2 = s1.clone();
        System.out.println("====");
        System.out.println("s2: "+s2.sid+".."+s2.sname+".."
                           +s2.addr.cityName+".."+s2.addr.stateName);
        System.out.println("====");
        s1.addr.cityName="chennai";
        s1.addr.stateName="tn";
        System.out.println("s1: "+s1.sid+".."+s1.sname+".."
                           +s1.addr.cityName+".."+s1.addr.stateName);
        System.out.println("====");
        System.out.println("s2: "+s2.sid+".."+s2.sname+".."
                           +s2.addr.cityName+".."+s2.addr.stateName);

    }
}
```

JAVA I/O (Input / Output) Streams:

Using Java application, we can store the data permanently in a specific place either in file or in database. If java application wants to communicate with file and database we need some low level predefine logic, that given by java software in two packages.

java.io

java.sql

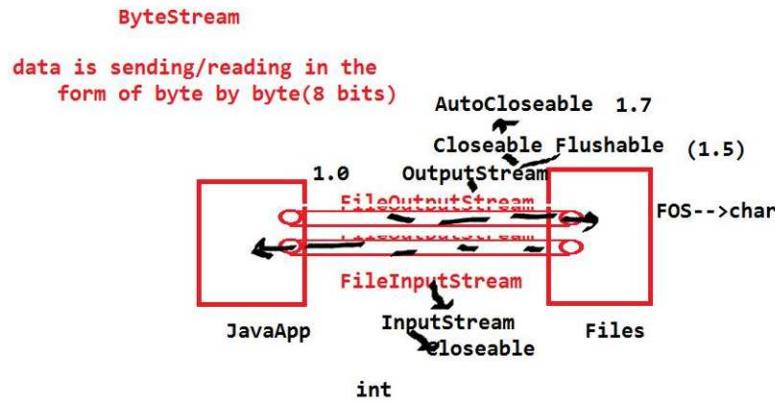
java. io package facilitates communication between java and files only.
java.sql package facilitates communication between java and different databases(oracle, my-sql, db2, mysql, sqlserver, tera-data etc) .The logic used to store data permanently in one place is called Persistent Logic and the location is called Persistent Store. or database.

Stream: The two-way data flow between java application and files is called stream.

In java, there are two types of streams.

- 1) Byte stream.
- 2) Character stream.

Data flow between java and files in form of byte by byte constitutes the Byte stream. Data flow between java and files in form of character by character constitutes the Character stream.



java.io.OutputStream:

`java.io.OutputStream` is the super class for all output stream classes under the byte stream for writing the data into files.

It is an abstract class. This abstract class implements two interface those are:

- 1) `java.io.Closeable`.
- 2)`java.io.Flushable`.

In this class we have one abstract method that is `write(int)`.

The various subclasses of this class they are:

`ByteArrayOutputStream`, `FileOutputStream`, `FilterOutputStream`, `ObjectOutputStream`, `PipedOutputStream`.

With the help of `FileOutputStream`, we can write data in byte format only. The data is stored in the file in the form character.

`FileOutputStream` uses either existing file or new file.

If file is not exists `FOS` created a new file with help given file name.

If file is existed FOS uses existed file name.

write() method and read() method always throws IOException. All io package classes throwing one more compile time exception that is java.io.FileNotFoundException.

FileInputStream is the basic class for reading the data from file.

This class always needs existed file name.

FIS class read the data from the file in the form of byte by byte.

The data is reading in the form int format.

```
import java.io.*;

public class FOSFISDemo{

    public static void main(String[] args) throws
FileNotFoundException,IOException{

        FileOutputStream fos = new FileOutputStream("vaibhav.text");

        fos.write('v');

        fos.write(65);

        fos.write('Z');

        System.out.println("data was entered in the form of character");

        FileInputStream fis = new FileInputStream("vaibhav.text");

        int i= 0;

        while((i=fis.read())!=-1){

            System.out.println(i+"..."+(char)i);

        }

        System.out.println("data was sucessfully read");

    }

}
```

Note: FileOutputStream fos = new FileOutputStream("vaibhav.text", true);

The above syntax allows new data to be appended to existing data, while keeping the old data intact.

java.io. InputStream:

java.io.InputStream is the super class for all the input stream classes under byte stream for reading data from file.

The various subclasses of this class are: AudioInputStream, ByteArrayInputStream, FileInputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream.

FileOutputStream and FileInputStream having the drawback like storing the data in the form of character and reading the data in the form of integer.

We cannot read the data in the different data type format.

To resolve this problem we can go for DataInputStream and DataOutputStream.

FOS,FIS are basic classes for writing and reading the data to and from the files.

All classes are using these two basic classes for communicating with the files.

```
package streams;

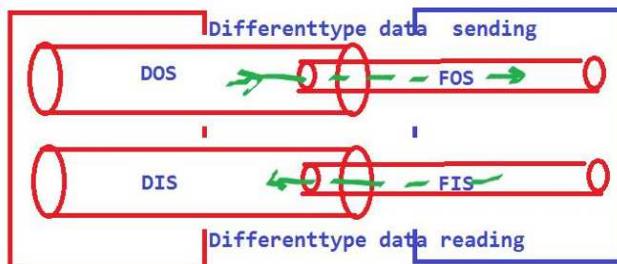
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class StreamDemo {
    public static void main(String[] args) throws FileNotFoundException,
IOException{
        try(FileOutputStream fos = new FileOutputStream("nit1")){
            byte b[] = {65,66,67,68,69};
            fos.write(b);
        }

        /*FileOutputStream fos = null;
        try{
        fos = new FileOutputStream("nit");
        byte[] b = {100,97,100};
        fos.write(b);
    }
}
```

```

    }
    catch(FileNotFoundException e){
        System.out.println("File is not available ");
    }
    catch(IOException e){
        System.out.println("writing and reading problem");
    }
    finally{
        try {
            if(fos !=null){
                fos.close();
            }
            else{
                System.out.println("connection is not established");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```



Whatever the order we insert the data into the file, in the same order we need read otherwise we will get unreliable result and exceptions



```

import java.io.*;

public class DOSDISDemo {

    public static void main(String[] args) throws
FileNotFoundException,IOException{
    FileOutputStream fos = new FileOutputStream("Vaibhav1.text");
    DataOutputStream dos = new DataOutputStream(fos);
    dos.writeByte(100);

    dos.writeChar('a');
}
}

```

```

dos.writeInt(120);

dos.writeBoolean(false);

FileInputStream fis = new FileInputStream("Vaibhav1.text");
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readByte());

System.out.println(dis.readChar());

System.out.println(dis.readInt());

System.out.println(dis.readBoolean());
}
}

```

Note: DataOutputStream and DataInputStream classes are not directly communicate with files, first these two classes are communicate with FileOutputStream and FileInputStream respectively, by using FOS and FIS functionalities DOS and DIS are communicating with files for writing and reading the data in the form of different datatypes.

ByteArrayOutputStream :-

```

import java.io.*;
public class ByteArrayDemo{
public static void main(String []args) throws FileNotFoundException {
    FileOutputStream fos1 = new FileOutputStream("siddharth1.text");
    FileOutputStream fos2 = new FileOutputStream("siddharth2.text");
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    bos.write(100);
    bos.writeto(fos1);
    bos.writeto(fos2);
    System.out.println("Success");
    FileInputStream fis1 = new FileInputStream("Siddharth1.text");
    FileInputStream fis2 = new FileInputStream("Siddharth2.text");
}
}

```

```

int i = 0;

while((i=sis.read()) != -1) {

    System.out.println(i+"...."+(char)i);

}

}

}

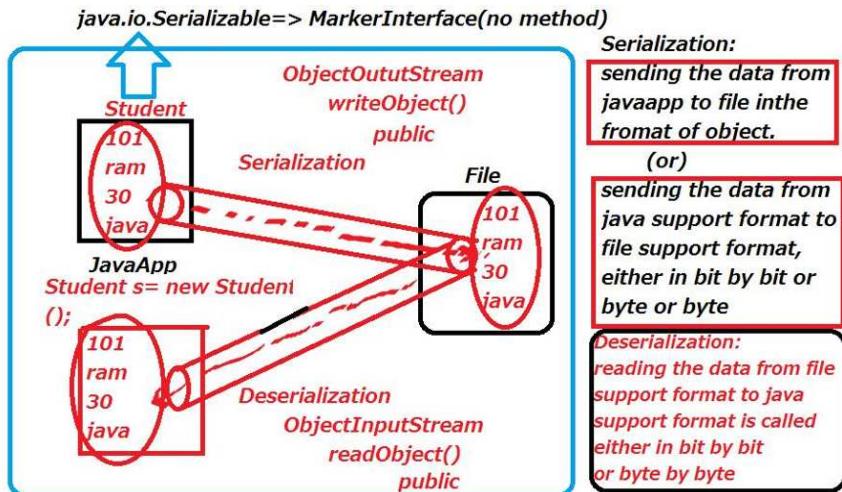
```

Serialization:

Converting/ translating data in the form of object from java supported architecture to file support architecture is called serialization. In the process of serialization, the data is encrypted according to java internal algorithms.

To support serialization, java provides pre-defined class -
`java.io.ObjectOutputStream(writeObject(obj))`.

It is mandatory that the object that is being serialized implements
`java.io.Serializable` interface (if not - `NotSerializableException`).



De-Serialization:

To translate the data from file support format to java support format is called de-serialization.

De-serialization is supported by pre-defined class
`java.io.ObjectInputStream(readObject(obj))`.

`readObject()` method will throws one compile time exception that is `ClassNotFoundException`.

In the deserialization jvm will create one new object, without executing the constructor.

java.io.Serializable:

It is a marker interface i.e it does not contain any methods. Any class implementing `Serializable`, JVM will treat that class as special object and thereby participates in serialization.

```
import java.io.*;

class Student implements Serializable{

    int sid = 111;      String sname = "siddharth";  int fee = 1000;
    String password = "kick";

}

public class SerializationDemo {

    public static void main(String []args) throws IOException,
    ClassNotFoundException

        FileOutputStream fos = new FileOutputStream("suresh.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        Student s = new Student();oos.writeObject(s);
        System.out.println("serialization success");

        FileInputStream fis = new FileInputStream("suresh.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Object obj = ois.readObject();

        Student s1 = (Student)obj;
        System.out.println(s1.sid);
        System.out.println(s1.sname);
```

```

        System.out.println(s1.fee);
        System.out.println(s1.password);
    }
}

}

```

Note:

To stop an object from participating in serialization, we should mention data as 'transient' with the keyword transient String password = "kick";
output -> NULL Static data is also does not participate in serialization.

```

static int sid = 111;
output -> 0

```

During Execution and in the process of deserialization, password is returned as NULL, but static value which is initially) is replaced by original value - 111.

Customized Serialization & De-serialization:-

```

import java.io.*;
class Student implements Serializable {
    int sid = 111;
    String sname = "siddharth";
    int fee = 1000;
    transient String password = "ramchandra";
private void writeObject(Object output stream oos) throws Exception {
    oos.default.writeObject();password = "123ramchandrahelohowru";
    oos.writeObject(password);
}
private void readObject(Object Input Stream ois) throws Exception {

```

```

        ois.default readObject();

        String password1 = (String)ois.readObject();

        password = password1.substring(3,13);

    }

public class SerializationDemo {

    public static void main(String []args) throws IO Exception,
ClassNotFoundException

    FileOutputStream fos = new FileOutputStream("suresh.ser");

    ObjectOutputStream oos = new ObjectOutputStream(fos);

    Student s = new Student();

    oos.writeObject(s);

    System.out.println("serialization success");

    FileInputStream fis = new FileInputStream("suresh.ser");

    ObjectInputStream ois = new ObjectInputStream(fis);

    object obj = ois.readObject();

    Student s1 = (Student)obj;

    System.out.println(s1.sid);

    System.out.println(s1.sname);

    System.out.println(s1.fee);

    System.out.println(s1.password);

}

}

}

```

Object Graph Serialization:-

```
class Dog implements Serializable {  
    Dog(){  
        System.out.println("zero argument constructor");  
    }  
    Cat c = new Cat();  
}  
  
Class Cat implements Serializable {  
    Rat r = new Rat ();  
}  
  
Class Rat implements Serializable {  
    int i = 111;  
}  
  
public class SerializationDemo {  
    public static void main(String []args) throws IO Exception, FilenotFound  
Exception {  
    /* instead of student object creation in the above main method, we create  
    dog object and pass it to write object */  
    Dog d = new Dog ();  
    oos.writeObject(d);  
    /*instead of casting to Student, we should cast into Dog*/  
    Dog d1 = (Dog)obj;  
    System.out.println(d1.c.r.i);  
}  
}  
}
```

Note:

In the above program, if any class (Dog/Cat/Rat) does not implement serializable interface, then we get a java.io.NotSerializable Exception.

Importance of order:

```
class Dog1 implements Serializable {  
    int i = 111;  
}  
  
Class Cat1 implements Serializable {  
    int j = 222;  
}  
  
Class Rat1 implements Serializable {  
    int k = 333;  
}  
  
public class SerializationDemo {  
    public static void main(String []args) throws IO Exception, FilenotFound  
        Exception {  
        FileOutputStream fos = new FileOutputStream("balaji.text");  
        ObjectOutputStream oos = new ObjectOutputStream(fos);  
        Dog1 d = new Dog1();  
        Cat1 c = new Cat1();  
        Rat1 r = new Rat1();  
        oos.writeObject(d);  
        oos.writeObject(c);  
        oos.writeObject(r);  
        System.out.println("Success");
```

```
FileInputStream fis = new FileInputStream("balaji.text");
ObjectInputStream ois = new ObjectInputStream(fis);
Rat1 r1 = (Rat1)ois.readObject();
Dog1 d1 = (Dog1)ois.readObject();
Cat1 c1 = (Cat1)ois.readObject();

/* the above order generates an error. Order is very important and order
given in serialization should be maintained for deserialization also. */
```

Serialization with Inheritance:

- If any sub-class undergoes serialization, then parent class is serialized automatically.

```
class Animal implements Serializable{
    int i = 111;
}

class Cow extends Animal implements Serializable {
    int j = 222;
}

(program as above)

Cow c = new Cow();
oos.writeObject(c);

(program as above)

Cow c1 = (cow)ois.readObject();
System.out.println(c1.i);
System.out.println(c1.j);
```

Note:

- If parent class implements Serializable then the sub class need not always implement Serializable.

```
class Animal implements Serializable{
```

```
    int i = 111;
```

```
}
```

```
class Cow extends Animal {
```

```
    int j = 222;
```

```
}
```

- If sub-class implements Serializable and the parent class does not implement Serializable, then the super class data does not participate in serialization.

```
class Animal {
```

```
    int i = 111;
```

```
}
```

```
class Cow extends Animal implements Serializable {
```

```
    int j = 222;
```

```
}
```

```
    Cow c = new Cow();
```

```
    c.j = 333;
```

```
    c.i = 444
```

```
output -> j = 333
```

```
i = 111 (not 444 because it did not participate in serialization)
```

Externalization:

```
class Student implements Externalizable {
```

```
    int sid;
```

```
String sname;
int sage;
public Student (){
    System.out.println("student constructor");
}
public Student (int sid, String sname, int sage){
    super();
    this.sid = sid;
    this.sname = sname;
    this.sage = sage;
}
public void readExternal(Object input ois) throws IO Exception,
ClassNotFoundException{
    String sname1 = (String)ois.readObject();sname = sname1;
}
public void writeExternal(object output oos) throws IO Exception,
ClassNotFoundException{
    sname = "krishna";
    oos.writeObject(sname);
}
}
public class Serialization Demo {
public static void main(String [] args) throws FileNotFoundException{
    Student s = new Student(101, "peri", 24);
    oos.writeObject(s);
    object obj = ois.readObject();
}
```

```

        Student s1 = (Student)obj;
        System.out.println(s1);
    }
}

```

- When reading data from file in the process of de-externalization, public zero argument constructor is mandatory. If not available - invalid class Exception.

Difference between Serialization & Externalization:

In Serialization, everything is taken care of by JVM (no user input) whereas in Externalization, everything is taken care of by programmer only.

Performance of serialization is low as compared to Externalization.

Serialization needs Serializable interface (marker interface - no method).

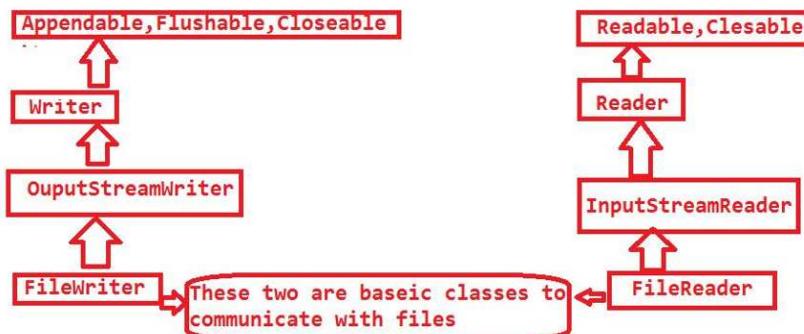
Externalization needs Externalizable interface.

Externalizable interface - needs two methods writeExternal() & readExternal().

Serialization does not require any default constructors whereas Externalization needs public zero argument constructor.

In Serialization there is a use of transient keyword but in externalization no use.

CharacterStreams



```
import java.io.*;
```

```

public class FWAFRDemo {
    public static void main(String[] args) throws
FileNotFoundException, IOException {
        FileWriter fw = new FileWriter("sony.text");
        fw.write(100);
        fw.write('a');
        fw.write('z');
        fw.flush();
        fw.close();
        FileReader fr = new FileReader("sony.text");
        int i=0;
        while((i=fr.read())!=-1){
            System.out.println(i+"...."+(char)i);
        }
    }
}

```

BufferedReader:

It is having two methods.

1. readLine():

Will read the data in the form of String.

2. read():

Will read only one single character ascii value. With the help of BufferedReader we can read the data from keyboard and also from files.

```

import java.io.*;
import java.util.StringTokenizer;

```

```
public class FWAFRDemo {  
    public static void main(String[] args) throws  
FileNotFoundException, IOException {  
    InputStreamReader isr = new InputStreamReader(System.in);  
    BufferedReader br = new BufferedReader(isr);  
    System.out.println("enter some data");  
    String s = br.readLine();  
    System.out.println(s);  
    System.out.println("enter some data");  
    int i = br.read();  
    System.out.println(i);  
    FileReader fr = new FileReader("sony.text");  
    BufferedReader br1 = new BufferedReader(fr);  
    String s1 = br1.readLine();  
    System.out.println(s1);  
    int total = 0;  
    StringTokenizer st = new StringTokenizer(s1, " ");  
    while(st.hasMoreTokens()){  
        total = total+Integer.parseInt(st.nextToken());  
    }  
    System.out.println(total);  
}  
}
```

java.io.Console:

```
import java.io.*;
class ConsoleDemo {
public static void main(String[] args)
throws FileNotFoundException,IOException {
    Console con = System.console();
    System.out.println("enter some data");
    String s = con.readLine();
    System.out.println(s);
    System.out.println("enter some data");
    char c[] = con.readPassword();
    System.out.println(c); String s1 = new String(c);
    System.out.println(s1);
}
class Check {
    public static void main(String...ram)throws Exception{
        System.out.println("check class");
        String s = System.getProperty("loadingclass");
        Class.forName(s);
    }
}
java -Dloadingclass=Student Check
```

System.out.println():

out is a static field in System class. So we are called out filed by using class name(System).

But println() is not available in System class, this println() available in java.io.PrintStream. It is a non-static method.

If we want call println() we need PrintStream class reference.

If we are creating reference PrintStream and calling println() the data is not print on the console, the data will be print on the file.

If we print the data on the console, we System.out. This System.out will represents output device like console.

If we are using System.out.println() then only the data will print on the console.

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
public class FWAFRDemo {
    public static void main(String[] args) throws
FileNotFoundException, IOException {
    FileOutputStream fos = new FileOutputStream("mona.text");
    PrintStream ps = new PrintStream(fos);
    ps.println("data is not print on console");
    System.out.println("data is print on console");
    System.err.println("this is also represents console");
    System.setOut(ps);
    System.setErr(ps);
    System.out.println("data is print on console");
```

```
        System.err.println("this is also represents console");
    }
}

class A implements Serializable{
    String s1 = "ram";
}
class B implements Serializable{
    String s2 = "suji";
}
class C implements Serializable{
    String s3 = "vani";
}
public class StreamDemo {
    public static void main(String[] args)
        throws FileNotFoundException, IOException,
        ClassNotFoundException{
        FileOutputStream fos = new FileOutputStream("file.text");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        A obj1 = new A();
        B obj2 = new B();
        C obj3 = new C();
        obj1.s1 = "java";
        obj2.s2 = "science";
        obj3.s3 = "SE";
        oos.writeObject(obj1);
        oos.writeObject(obj2);
        oos.writeObject(obj3);
        FileInputStream fis = new FileInputStream("file.text");
        ObjectInputStream ois = new ObjectInputStream(fis);

        for(int i=0;i<3;i++){
            Object o = ois.readObject();
            if(o instanceof A){
                A obj4 = (A)o;
                System.out.println(obj4.s1);
            }
            else if(o instanceof B){
                B obj5 = (B)o;
                System.out.println(obj5.s2);
            }
            else{
                C obj6 = (C)o;
                System.out.println(obj6.s3);
            }
        }
    }
}
```

MULTI -THREADING

MultiTasking:

Processing/Executing more than one task /several tasks simultaneously is called MultiTasking.

There two types of multitasking. They are:

Process based multitasking.

Thread based multitasking.

Processed Based Multitasking:

Processing multiple tasks simultaneously based on address/process is called process Based Multitasking.

Here each job can be done by separate process.

Example:

The best example of Process Based Multitasking is when we reading a book first we see the word after we are reading/speaking the word.

Here seeing the word is one task and reading word is another task.

These two tasks can be done by simultaneously.

In the Process Based Multitasking switching one context /area to another context/ area is take more time.

Each and every process having its own address/context, so switching from one context to another context takes more time. It is heavy weight process.

Resources consuming is more.

Thread Based Multitasking:

Processing multiple jobs/tasks simultaneously based on Thread is call Thread Based Multitasking.

Here each task is separately individual part.

This is best suitable for programming.

Thread uses less resources when compare Process.

We simply say this; collection of threads is called process.

Here all threads are placed in a one context/ area, so switching the control from one context to context is not take that much of time when compare to Process switching.

That's why comparing process based multitasking, thread based multitasking best suitable for application performance.

We can also say thread based multitasking as Multithreading.

In the both cases of multitasking we should get performance, the reason is reducing the time that means complete the task quickly. This is light weight process. Resource consuming is less. Java provides huge libraries to work with multi threading in the form of API like Runnable, Thread, ThreadGroup.

Thread:

A thread represents a separate path of execution of a group of statements.

In java program, we have group of statements, these are executed one by one statements by JVM. We can also specify Thread is a stack which is created in Java Stacks Area. In our program the default thread is main, which is executed by JVM.

(Q) Why should we go for Multithreading?

Threads are mainly used in server-side programming.

Threads can give the response to all the clients with in the same time.

Threads can also used at developing games, animation.

In java simply we can say JVM is a process, which is by default having two Threads for each and every program. They are:

main: Used to execute the methods.

garbage collector: Deleting the objects, which are not used by using mechanism called Mark and Sweep.

When we working with threads we can see two types of execution, they are:

Sequential execution:

A single thread can executes each and every method one by one. In sequential execution the time for the completion of work is more.

Concurrent execution: Methods are executed simultaneously. The completion of the work is very fast when compare to sequential execution.
Note: At a time a thread can execute only one time.

In concurrent execution the tasks will be processed in the bellow manner.

Starts-suspend-resume-end.

In this process user should create multiple threads to work with multiple tasks.

```
public class Demo {  
    void m1(){  
        System.out.println("m1-method: "+  
                           Thread.currentThread().getName());  
    }  
    static void m2(){  
        System.out.println("m2-method: "+  
                           Thread.currentThread().getName());  
    }  
    public static void main(String[] args) {  
        System.out.println("main-method: "+
```

```

        Thread.currentThread().getName());
Demo d = new Demo();
d.m1();
m2();
}
}

```

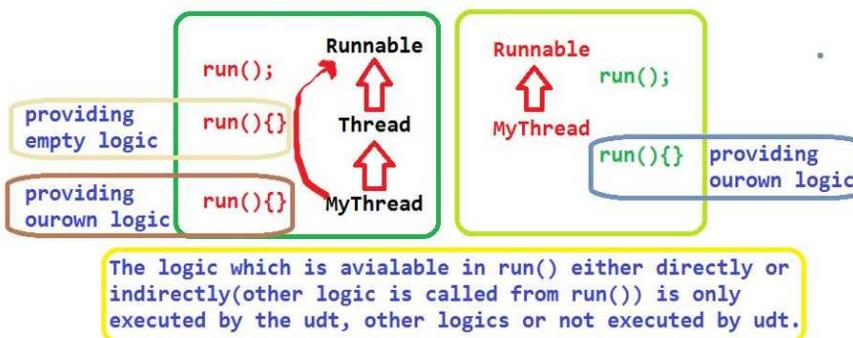
In the above program we unable to get multithreading concept.

To develop multithreading concept we required multiple threads.

Creation of UserDefine/Custom Thread:

If we want to create user define thread in java we have two ways.

- By extends `java.lang.Thread` class
- By implements `java.lang.Runnable` interface.



If we want to assign any task to user define thread, first we need to place that work in run method either directly or indirectly.

directly:

```

public void run(){
    //writing logic
}

```

Note: If we are calling run() by directly in main(), again that run() will be call by main thread only.
So If We Want to execute run() by userdefine thread, we should use start()

indirectly:

```

public void run(){
    //calling method
    m1();
}

```

```

void m1(){
    //logic
}

```

By extends java.lang.Thread:

```
package thread;

public class MyThread extends java.lang.Thread{

    public void run(){

        System.out.println("varun/userdefine thered" +
            " priority: "+Thread.currentThread().getPriority());

        for(int i =0; i< 10; i++){

            System.out.println(Thread.currentThread().getName()+
                ".."+i);

        }

    }

    public static void main(String[] args) {

        MyThread mt = new MyThread();

        //mt.run();

        mt.setName("varun");

        //mt.setPriority(10);

        //mt.setPriority(-11);

        mt.start();

        //mt.start();

        //mt.stop();

        System.out.println("main thered priority: "+
            Thread.currentThread().getPriority());

        for(int i =0; i< 10; i++){

            System.out.println(Thread.currentThread().getName()+
```

```
        .."+i);

        //Thread.currentThread().stop();

    }

    System.out.println(Thread.MIN_PRIORITY);

    System.out.println(Thread.NORM_PRIORITY);

    System.out.println(Thread.MAX_PRIORITY);

}

}
```

First we should take one class, that class must be extends java.lang.Thread class. The reason if we want develop any User define thread class, that class must be need java.lang.Thread class functionalaties.

```
class MyThread extends java.lang.Thread{

}
```

If we want execute user define logic with the help of userdefine thread, that code must be write within the run().

All the main method statements execute by the main thread.

If we are calling any method from main(), that called method statements also execute by the main thread only.

So if we are calling run () directly, then main thread will be execute that method but not user define thread.

If we want execute run () by the user define thread then weshould we start().

start () of Thread class automatically calls user define override run(). That means that run () internally execute by user define thread only.

Every logic(method) executes by the main thread only. If we want to execute any logic by the user define thread, that logic must be place in the run method.
If we are calling run() directly that is also executing by main thread. so if we want to execute run() by the user define thread we need to call start(), with the help of this start() logic only our class will treated as user define thread class and by using that user define thread class jvm will executes run().

```
class Thread {  
    public synchronized void start(){  
        -----//register our class with  
        -----//thread class libraries and making our class  
        -----//as an user define thread  
        this.run();  
    }  
}
```

If we want start the thread we need start().

If we want stop the thread we have stop().

If we want to know the thread name

Then we should bellow syntax.

Thread.currentThread().getName().

If we want to give name to thread, we have method like

setName(String name).

Thread priorities are start from 1 to 10.

Main thread priority is 5. Whatever the threads which are created by the main thread those thread priority is also 5.

Java provides three constants to know the priority of an threads those are

MIN_PRIORITY

NORM_PRIORITY

MAX_PRIORITY

Thread minimum priority is 1.

Normal priority is 5

Maximum priority is 10.

If we want setting priority and getting priorities of a thread we have two predefine methods.

1.setPriority()

2.getPriority()

If we are setting thread priorities more than 10 and Less than 1 then we will get runtime exception this is Java.lang.IllegalArgumentException.

We cannot call start () more than one time on one single thread object.

If we are calling start () more than one time, then we will get one runtime exception java.lang.IllegalThreadStateException.

Thread execution is not reliable, thread execution entirely depend upon JVM.

Q) Can we override start () in our class?

A) Yes. We can.

```
public class Demo extends Thread {  
  
    @Override  
    public void run(){  
        System.out.println("run_method: "+  
                           Thread.currentThread().getName());  
    }  
  
    @Override  
    public void start(){  
        System.out.println("start_method: "+  
                           Thread.currentThread().getName());  
        run(); //this.run()  
    }  
  
    public static void main(String[] args) {  
        System.out.println("main_method: "+  
                           Thread.currentThread().getName());  
        Demo d = new Demo();  
        //d.run();  
        d.setName("kit");  
        d.start();  
    }  
}
```

If we calling start () on user define thread object, then control not goes to start () of Thread class. User define Start () will be execute. In this time run () not executed.

If we are overriding the start () method, that method will be treated as a normal method by the JVM.

Note: If we want to call the run () by the user define thread, then we should forward our control to java.lang.Thread class start ().

Creating user define thread by implementing java.lang.Runnable interface

Program: -----need to add----

Q)Difference between extends Thread implements Runnable?

A)If we extends Thread, we are unable to extends some other class functionalities, the reason java does not support multiple inheritance through classes.

But if we go for Runnable interface we can access Runnable functionalaties and some other class funcationalities.

If we extends Thread class, we are unable use same object more than one time. If we call start() on same object more than one time we will one exception that is java.lang.IllegalThreadStateException.

But if we implements Runnable, we can same object more than one with the help of Thread class reference.

If we extends Thread class we will get all functionalites, but if we go for Runnable we will get required functionality.

extends Thread	implements Runnable
If we extends Thread, our class unable extends some other class, the reason java doesn't support multiple inheritance.	If we implements Runnable, we can make our class as an thread class and also we will have some other class functionalities.
If we extends Thread, our reference can not be reusable.	If we implements Runnable, we reuse our user define thread reference.
If we extends Thread, we will have all functionalities.	If we want access limited functionalities, we should implements Runnable

Controlling the threads:

1. yield ():

If we want forwarding the control from one thread to another thread based on priority then we can go for yield().

Before giving the control to waiting thread, currently executing thread checks priority of waiting thread.

If waiting thread priority is greater than or equal to current thread then automatically control goes to waiting thread.

2. join(): If we want push the waiting thread into execution state then we can use join(). Java provide three types of join methods.

join()

join(-)

join(-,-)

Q) What is difference between join() and join(-)?

A) If we are using join(), then joined thread will execute entire work. Whereas join(-) will give chance to thread only some period of time.

Within the period of time if work is completed then control goes to waiting thread, otherwise in the middle of the work only control goes to waiting thread.

3.sleep():

This method is used to placing the current execution thread in sleep mode for particular time. We have two different sleep methods.

sleep(-) and sleep(--)

Note: Both join() and sleep() are throwing compile time Exception. That is java.lang.InterruptedException

Differences between join and sleep and yield methods.

	Yield	join	sleep
Number of method:	1	3	2
Static	yes	no	yes
Native	yes	no	yes
Final	no	yes	no
Interrupted	no	yes	yes
Exception			
Synchronized	no	2—syn(join(-) no Join(--) Join()(not synt)	

Synchronization:

It is a mechanism, which is used to allow only one thread to execute the entire resource.

If more than one thread is executing the only one common resource, then we should allow only one thread at a time. After complete the work by the allowed thread, then remaining threads will get chance to execute.

If we want to get these type of functionalities, we should go for synchronization.

If we want to achieve synchronization in java, we have one keyword that is "synchronized".

"synchronized" keyword we can be applied on top of methods and blocks not on classes and variables.

If more than one thread executing the single resource we will get inconsistency output. (Race condition)

If any one thread executing the synchronized method, first that thread will get lock on that object, once that method execution completed thread will releases the lock. Getting the lock and releasing the lock is taken care by jvm only.

Jvm will give the chance to only one thread to execute the synchronized method. Once thread is executing the synchronized method, remaining threads can not execute same synchronized method, but other threads can execute other non-synchronized methods.

```
class Cinema{
    synchronized public void cinemaTalk(String name){
        for(int i=0;i<10;i++){
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("good: "+name);
        }
        System.out.println("=====");
    }

    void m1()throws InterruptedException{
        for(int i=0;i<10;i++){
            Thread.sleep(100);
            System.out.println("m1 method"+
            Thread.currentThread().getName());
        }
    }
}
```

```

class Person extends Thread{
    Cinema c;
    String name;
    Person(Cinema c , String name ){
        this.c = c;
        this.name = name;
    }
    public void run(){
        c.cinemaTalk(name);
    }
}
public class Synchronization extends Thread{
    public static void main(String[] args)
        throws InterruptedException{
    Cinema c = new Cinema();//1010
    Person p1 = new Person(c,"ram");
    p1.start();

    Person p2 = new Person(c,"sam");
    p2.start();
    c.m1();
}
}

```

Ex:

```

synchronized void m1 () {
    //synchronized method
}

Void m2 () {
    synchronized (MyThread.class){
        //synchronized block
    }
}

```

```

class Display{
    synchronized void wish(String msg){
        for(int i=0;i<10;i++){

```

```
        System.out.println("good morning " +msg);

    }

try{
    Thread.sleep(5000);
}catch(Exception e){}
}

}

class MyThread1 extends Thread{

    Display d1;
    String msg;
    MyThread1(Display d1, String msg){
        this.d1 = d1;
        this.msg=msg;
    }
    public void run(){
        d1.wish(msg);
    }
}

public class SynchronizedDemo {
    public static void main(String[] args) {
        Display d = new Display();
        MyThread1 m1 = new MyThread1(d,"ram");
        MyThread1 m2 = new MyThread1(d,"sam");
    }
}
```

```
    m1.start();  
    m2.start();  
}  
}
```

In the following scenario we have two threads on two different resources, that means every thread has its own resource, in that time if we are applying synchronization there is no usage.

```
class Display{  
  
    synchronized void wish(String msg){  
  
        for(int i=0;i<10;i++){  
  
            System.out.println("good morning " +msg);  
  
        }  
  
        try{  
  
            Thread.sleep(5000);  
  
        }catch(Exception e){}  
  
    }  
}
```

```
class MyThread1 extends Thread{  
  
    Display d1;  
  
    String msg;  
  
    MyThread1(Display d1, String msg){  
  
        this.d1 = d1;  
  
        this.msg=msg;  
    }
```

```

}

public void run(){
    d1.wish(msg);
}

}

public class SynchronizedDemo {
    public static void main(String[] args) {
        Display d = new Display();
        Display d1 = new Display();
        MyThread1 m1 = new MyThread1(d,"ram");
        MyThread1 m2 = new MyThread1(d1,"sam");
        m1.start();
        m2.start();
    }
}

```

Difference between join() and synchronization?

Whenever we call join() on top of one thread, then join thread will executing the task, current thread will be in the stop mode.
 If we are allowe two thread on synchronized method at a time one thread will execute task, remaining thread is not in stop mode, can execute other work.

When ever we call join() on top of any thread that thread will execute entire task(all statements), but with the help of synchronization block we execute some of the statements of the method by only one thread at a time.

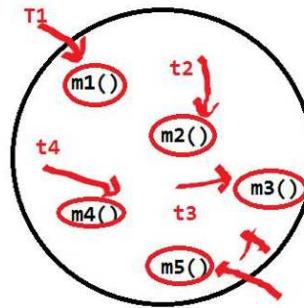
Performance: Both low, but comparing to join(), synchronization having little bit of more performance.

If any thread which executes static synchronized methods, remaining threads not execute static synchrnozed method but remaining thread can executes

static non-synchronized methods
 non-static non-synchronized methods
 non-static synchronized methods.

```

class SynDemo{
    static synchronized void m1(){}
    static void m2();
    synchronized void m3(){}
    void m 4();
    static synchronized void m5(){}
}
  
```



In this particular time we should go for locking mechanism.

We have two types locking.

1. Class level locking.
2. Object level locking.

```

class DemoClass{

    public void demoMethod(String str){

        //synchronized(this){ //object level locking

        synchronized(DemoClass.class){ //class level locking

            for(int i=0;i<10;i++){

                System.out.println("good morning"+..."+"+str);

            }

            try{Thread.sleep(2000);

            }catch(Exception e){

                e.printStackTrace();

            }

        }

    }

}
  
```

```
        }

    }

}

class Supported extends Thread{

    DemoClass d;
    String str;

    Supported(DemoClass d,String str){

        this.d =d;
        this.str = str;
    }

    public void run(){

        d.demoMethod(str);
    }
}

public class SynchronizedDemo1 {

    public static void main(String[] args) {

        DemoClass dc = new DemoClass();
        DemoClass dc1 = new DemoClass();
        Supported s =  new Supported(dc,"ram");
        Supported s1 =  new Supported(dc1,"sam");
        s.start();
        s1.start();
    }
}
```

```
 }  
}
```

Drawbacks: Performance will be decreases.

The reason is multiple threads are in the waiting mode, program execution will take more time, if execution time is increases automatically performance will be decreases.

But synchronization is very good at result. Provides accurate/reliable results.

resume() and suspend():

With the help of suspend() we can stop the particular Thread.

With the help of resume() we can start the particular thread.

Inter Thread communication:

```
class Customer{  
    int amount=10000;  
  
    synchronized void withdraw(int amount){  
        System.out.println("going to withdraw...");  
        if(this.amount<amount){  
            System.out.println("Less balance; waiting " +  
                "for deposit...");  
        try{  
            //wait(5000);  
            wait();  
        }catch(Exception e){  
        }  
    }
```

```
        }

        this.amount=this.amount-amount;

        System.out.println("After withdrawl: "+this.amount);

        System.out.println("withdraw completed...");

    }

    synchronized void deposit(int amount){

        System.out.println("going to deposit...");

        System.out.println("before deposit: "+this.amount);

        this.amount = this.amount+ amount;

        System.out.println ("depositcompleted... ");

        System.out.println("After Deposit: "+this.amount);

        notifyAll();

        //notify();

    }

}

public class WaitTest{

    public static void main(String args[]) throws InterruptedException{

        int a = 10/0;

        final Customer c=new Customer();

        new Thread(){

            {

                public void run(){


```

```

        c.withdraw(15000);

    }

}.start();

new Thread(){

    public void run(){

        c.deposit(10000);

    }

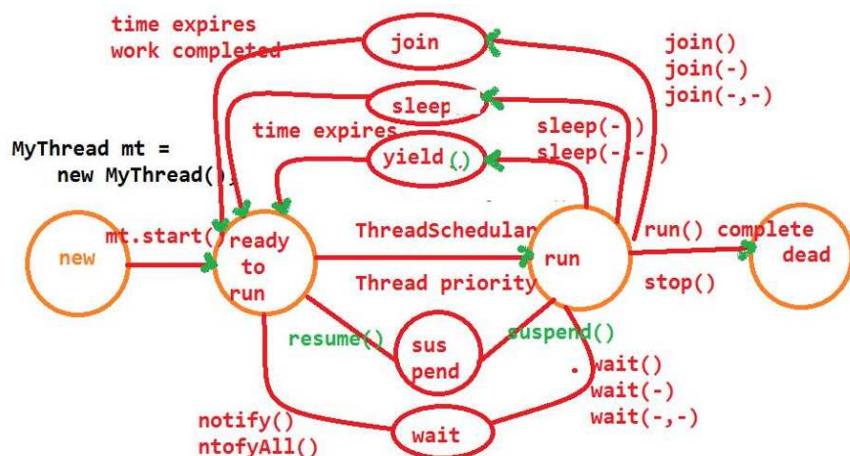
}.start();

}

}

```

Thread Life Cycle:



Before executing run() by the userdefine thread, whatever priority to set that user define thread , that priority only applicable to that that thread.

```

class MyThread extends Thread{
    public void run(){

        try {
            Thread.sleep(3000);

```

```

    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    System.out.println("run : "+
        Thread.currentThread().getName());
    System.out.println("run : "+
        Thread.currentThread().getPriority());
}
public static void main(String[] s) throws InterruptedException{
    MyThread mt = new MyThread();
    MyThread mt1 = new MyThread();

    mt.setName("ram");
    //mt.setPriority(11); //java.lang.IAE
    mt.setPriority(9);
    mt.setPriority(4);
    mt1.setName("yaane");
    mt1.start();

    mt1.join();
    mt.start();
    mt.setPriority(6);
    //mt.start();
}

}

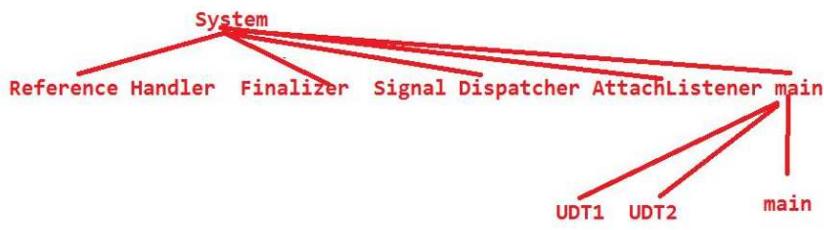
```

Garbage Collection:

In java we have two types of threads.

1. Daemon Thread.
2. Non-Daemon Thread.

We have following 5 import threads.



Thread: Executing group of statement in the separate path.

Non-Daemon Thread:

These are high priority thread.

These threads must be executed.

Ex: main thread.

All user define threads also come under Non-Daemon thread.

The default value of Non-Daemon thread is '5'.

If we want to check whether the thread is daemon or not, in java we have one method that is "isDaemon()".

If thread is Daemon then this method returns true, otherwise returns false.

We can change our thread from non-daemon to daemon, with the help of one method i.e "setDaemon".

Daemon Thread:

These are low priority threads.

We are not given guarantee to execute.

ex: Garbage collector.

It is an one low priority thread.

It is executed in the background by the jvm.

```
public class MyThread extends Thread{
    public void run(){
        System.out.println("udt thread type: "+
                           Thread.currentThread().isDaemon());
        try{
            for(int i=0;i<1000;i++){
                Thread.sleep(100);
                System.out.println("run: "+
                                   Thread.currentThread().getName()+" "+i);
            }
        } catch(InterruptedException e){
        }
    }
    public static void main(String[] args) {
        System.out.println("main thread type: "+
                           Thread.currentThread().isDaemon());
        MyThread mt = new MyThread();
        mt.setName("yaane");
        mt.setDaemon(true);
        mt.start();
        for(int i=0;i<1000;i++){
            System.out.println("main: "+
                               Thread.currentThread().getName()+" "+i);
        }
    }
}
```

Why should we go for GarbageCollection?

If we go to any other language like c++, if we want provide memory(allocating memory) our data programmer should write some code and also if we want to deallocating the memory in this time programmer should write some code.

If programmer forgot about deallocating memory code, some time later the memory will be filled, there may be chance of program is going to be stop .

To avoid above drawback java introduced one predefine service or functionalaty i.e memory deallocation with the help of Garbagecollector thread.

To cleanup the unused memory is called GarbageCollection.

```
public class Test {  
    static void m1(){  
        System.out.println("m1 method");  
        Test t1 = new Test();  
        Test t2 = new Test();  
    }  
    public static void main(String[] args) {  
        Test t = new Test();  
        Test t1 = new Test();  
        Test t2 = new Test();  
        //no object garbaged  
        t1=null;  
        //one object  
        t=null;  
        //two object  
        t2= null;  
        //three object  
        m1();  
        System.out.println("*****");  
    }  
}
```

Communicating with GarbageCollector:

In java we have two to communicate the garbage collector.

1. System.gc();
2. Runtime.getRuntime().gc();

GarbageCollector internally calls finalize() method to deallocate memory.

```
public void finalize(){  
}
```

Another name of Garbage Collector is Finalizer.

Executing the finalize method and deallocating the memory is called finalization.

```
public class Test {  
  
    public void finalize(){  
  
        System.out.println(Thread.currentThread().getName()+" object garbaged");  
  
    }  
  
    public static void main(String[] args) {  
  
        Test t = new Test();  
  
        t=null;  
  
        System.gc();  
  
        Test t1 = new Test();  
  
        t1.finalize();  
  
        Test t2 = new Test();  
  
        t2 = null;  
  
        //Runtime rt = new Runtime(); //wrong  
  
        Runtime rt1 = Runtime.getRuntime();  
  
        rt1.gc();  
  
        //System.gc();
```

```
}
```

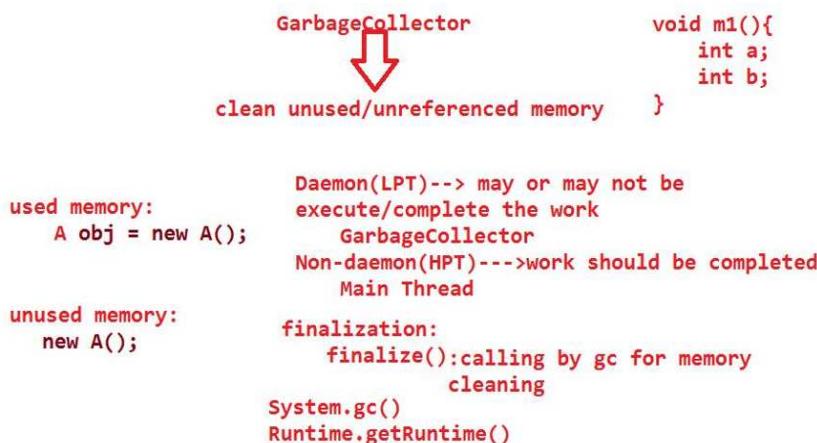
```
}
```

If we are calling finalize() explicitly like object.finalize(), in this time Garbage collector not execute finalize method, the respective thread will be execute the finalize(). So it is just method calling not finalization.

```
final: keyword
    apply on variable,methods,class
    final variable not change
    final method not participated in
        Method overriding
    final class not participated in
        inheritance

finally: block
    holding resource cleanup code
    closing db connection,closing file
    connection

finalize(): method
    it is calling by gc to clean the
    memory
```



If finalize() internally calls by garbagecollector then only memory will be deallocated otherwise memory not deallocated.

If we are calling System.gc() and Runtime.getRuntime().gc() repectively only one time memory will be garbaged.(not more than one time)

```
public class Test {
    int i = 111;
```

```
public static void main(String[] args) {  
    Test t1 = new Test();  
    Test t2 = new Test();  
    System.out.println("t1: "+t1.i);  
    System.out.println("t2: "+t2.i);  
    t2.i=333;  
    t1=t2;  
    System.out.println("t1 object is garbaged");  
    System.out.println("t1: "+t1.i);  
    System.out.println("t2: "+t2.i);  
    t1.i=999;  
    System.out.println("t1: "+t1.i);  
    System.out.println("t2: "+t2.i);  
}  
}  
  
public class Test {  
    int i = 111;  
    static void m1(Test t1){  
        Test t2 = new Test();  
        Test t3 = new Test();  
    }  
    public static void main(String[] args) {  
        Test t = new Test();  
        System.out.println(t.i);  
    }  
}
```

```
    m1(t);  
    System.out.println(t.i);  
}  
}
```

In the above program t2 and t3 memory will be deallocated and t1 reference will cancelled.

```
public class Test {  
    public void finalize(){  
        System.out.println(Thread.currentThread().getName()  
            +" is called");  
        int a = 10/0;  
    }  
    public static void main(String[] args) {  
        Test t = new Test();  
        //t.finalize();  
        t = null;  
        System.gc();  
    }  
}
```

When ever Finalizer(GarbageCollector) calls the finalize(), if any

RuntimeExceptions are raised those are not thrown by finalize() that means unchecked exceptions are not thrown by finalize().

but if any thread call finalize() explicitly then finalize() will throws uncheckedexception messages.

```
public class MyThread extends Thread{
```

```
static Thread mt;

public void run(){
    try{
        mt.join();
    }
    catch(InterruptedException e){
        e.printStackTrace();
    }
    for(int i=0;i<10;i++){
        System.out.println("child thread");
    }
}

public static void main(String[] args) throws InterruptedException,
InstantiationException, IllegalAccessException {
    MyThread.mt = Thread.currentThread();
    MyThread mt1 = new MyThread();
    mt1.start();
    for(int i=0;i<10;i++){
        System.out.println("main thread");
        Thread.sleep(1000);
        mt1.join();
    }
}
```

```
}

=====
public class MyThread extends Thread {

    @Override
    public void run(){
        for(int i=1;i<=10;i++){
            System.out.println("run: "+" "+i+" "+
                Thread.currentThread().getName());
        }
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

/*static{ //invalid
    System.out.println("static block");
    Thread t = Thread.currentThread();
    System.out.println("*****: "+t.getName());
    t.setDaemon(true);
}*/



public static void main(String[] args){
    /*Thread t = Thread.currentThread(); //invalid
    System.out.println("*****: "+t.getName());
```

```
t.setDaemon(true);*/  
MyThread mt = new MyThread();  
mt.setName("UDT");  
System.out.println("check: "+mt.isDaemon());  
mt.setDaemon(true);  
System.out.println("check: "+mt.isDaemon());  
mt.start();  
//mt.setDaemon(true);  
//dont use setDaemon() after start() method  
for(int i=1;i<=10;i++)  
    System.out.println("main: "+" "+i+" "+  
                      Thread.currentThread().getName());  
}  
}  
  
=====
```

```
ThreadGroup tg = Thread.currentThread().getThreadGroup().getParent();  
System.out.println(tg.getName());  
Thread[] t = new Thread[tg.activeCount()];  
System.out.println(t.length);  
tg.enumerate(t);  
for(Thread t1: t)  
{  
    System.out.println(t1.getName());
```

```
}

=====
package threads;

public class MyThread extends Thread {

    MyThread(ThreadGroup tg, String name){
        super(tg, name);
    }

    public void run(){
        System.out.println("child thread");
        try{
            Thread.sleep(5000);
        }
        catch(Exception e){
        }
    }

    public static void main(String[] args) throws InterruptedException{
        ThreadGroup tg = new ThreadGroup("ParentGroup");
        ThreadGroup cg = new ThreadGroup(tg, "childGroup");
        MyThread mt1 = new MyThread(tg, "childthread1");
        MyThread mt2 = new MyThread(tg, "childthread2");
        mt1.start();
        mt2.start();
        /*MyThread mt3 = new MyThread(cg, "childthread3");
        mt3.start();*/
```

```
        System.out.println("tg*****:"+tg.activeCount());
        System.out.println("tg1*****:"+tg.activeGroupCount());
        System.out.println("cg*****:"+cg.activeCount());
        System.out.println("cg1*****:"+cg.activeGroupCount());
    }

}

=====
package threads;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class BookTicket implements Runnable{

    String name;
    BookTicket(String name){
        this.name = name;
    }

    public void run(){
        System.out.println("Ticket is trying to book by:
"+name+"...."+Thread.currentThread().getName());
        try{
            Thread.sleep(1000);
        }
        catch(Exception e){
    
```

```
    }

    System.out.println("Ticket was booked by:
"+name+"....."+Thread.currentThread().getName());

}

}

public class MyThread{

    public static void main(String[] args) throws InterruptedException{

        BookTicket bt1 = new BookTicket("ram");

        //new Thread(bt1).start();

        BookTicket bt2 = new BookTicket("sam");

        //new Thread(bt2).start();

        BookTicket bt3 = new BookTicket("kiran");

        //new Thread(bt3).start();

        BookTicket bt4 = new BookTicket("varun");

        //new Thread(bt4).start();

        BookTicket bt5 = new BookTicket("suji");

        //new Thread(bt5).start();

        BookTicket[] bt11 = {bt1,bt2,bt3,bt4,bt5};

        ExecutorService es = Executors.newFixedThreadPool(3);

        for(BookTicket bt12: bt11){

            es.submit(bt12);

        }

        es.shutdown();

        /*BookTicket[] bt1 = {new BookTicket("ram"),new
BookTicket("sam"),new BookTicket("kiran"),
```

```
        new BookTicket("varun"),new BookTicket("suji")};

    ExecutorService es = Executors.newFixedThreadPool(3);

    for(BookTicket bt2: bt1){

        es.submit(bt2);

    }

    es.shutdown();*/
}

=====

package threads;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

class BookTicket implements Callable{

    int count;

    BookTicket(int count){

        this.count = count;

    }

    @Override

    public Object call() throws Exception {

        int sum=0;
```

```
System.out.println(Thread.currentThread().getName()+
    ": is trying to calculate sum ");

for(int i=0;i<=count;i++){
    sum=sum+i;
    i++;
}

return sum;
}

public class MyThread{

    public static void main(String[] args) throws InterruptedException,
ExecutionException{
        BookTicket bt1 = new BookTicket(10);
        BookTicket bt2 = new BookTicket(20);
        BookTicket bt3 = new BookTicket(30);
        BookTicket bt4 = new BookTicket(40);
        BookTicket bt5 = new BookTicket(50);
        BookTicket[] bt11 = {bt1,bt2,bt3,bt4,bt5};
        ExecutorService es = Executors.newFixedThreadPool(3);
        for(BookTicket bt12: bt11){
            Future f =es.submit(bt12);
            System.out.println(f.get());
        }
        es.shutdown();
    }
}
```

```
}

=====

class ThreadLocalDemo{

    ThreadLocalDemo(){
        ThreadLocal tl = new ThreadLocal();
        System.out.println(tl.get());
        tl.set(123);
        System.out.println(tl.get());
        tl.remove();
        System.out.println(tl.get());
    }
}

class ThreadLocalDemo1{

    ThreadLocalDemo1(){
        ThreadLocal tl = new ThreadLocal(){
            public Object initialValue(){
                return "ram";
            }
        };
        System.out.println(tl.get());
        tl.set("sam");
        System.out.println(tl.get());
        tl.remove();
    }
}
```

```
        System.out.println(tl.get());
    }
}

public class MyThread{
    public static void main(String[] args){
        new ThreadLocalDemo();
        new ThreadLocalDemo1();
    }
}

=====
class ThreadLocalDemo extends Thread{
    static Integer id=0;
    ThreadLocal tl = new ThreadLocal(){
        protected Object initialValue(){
            return ++id;
        }
    };
    ThreadLocalDemo(String name){
        super(name);
    }
    public void run(){
        System.out.println(Thread.currentThread().getName()+" id is:
"+tl.get());
    }
}
```

```

}

public class MyThread{
    public static void main(String[] args){
        ThreadLocalDemo tld1 = new ThreadLocalDemo("ram1");
        ThreadLocalDemo tld2 = new ThreadLocalDemo("ram2");
        ThreadLocalDemo tld3= new ThreadLocalDemo("ram3");
        ThreadLocalDemo tld4 = new ThreadLocalDemo("ram4");
        tld1.start();
        tld2.start();
        tld3.start();
        tld4.start();
    }
}

```

COLLECTION FRAME WORK

Drawbacks of Arrays:

- Arrays are fixed in size.
 - int a [] = new a [10]
 - Here "10" is size of an array.
- Once we creating an array there is no chance of increasing/decreasing its size based on our requirement.
- Array can hold only homogeneous data elements.

- Teacher t[] = new Teacher[10];
- Here we can add only teacher object, but it cannot hold student, employee objects data.
- If we want overcome this problem we should go for Object array.
 - Object o[] = new Object[10];
 - o[1] = "rams"
 - Here "rams" is an string object
 - o[2] = new Teacher();
 - Here we did place the Teacher object.
- Arrays can allow the duplication code/data.
 - int a[] = {10, 20, 6, 34, 10}
- Arrays don't provide any low level services.
- Arrays don't have any searching, sorting logic by default; we have to write the entire low level code/functionalities.
- To resolve these above problems we should go for collections framework.

1. Primitive variable can hold only single value.

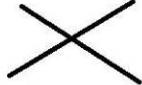
2. Reference variable can hold multiple values but only one object at a time.
array

3. double[] d = new double[1];
d[0] = false; //cannot hold incompatible data.

4. Student[] s = new Student[3];
s hold multiple values and multiple memories of student type but not hold other types.

5. There is no default sorting technique.
Like Ascending order
Descending order
LIFO
FIFO

6. There is no searching technique.



Whenever we placing data in the middle of an array I want shift my elements to right side, but it is not possible.

There is no simple way to delete the elements from array.

Without replacing old data with new data we need place the data into array, it is not possible.

array size is fixed.

array does not avoid duplicates

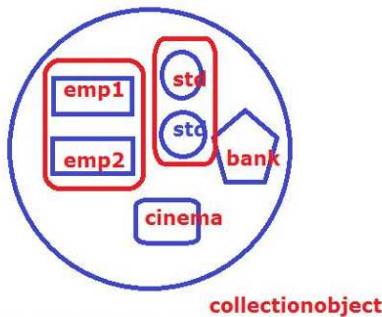
we want restrict to array to read the only in sequence order.

Instead of sending data individually from one class to another, if we want to send all the data as a single method parameter or method return type at a time then we should go for collection.

Operation on Collection Object:
=====

1. adding (add)
2. searching (indexOf(Object))
3. Number of elements(count)(size())
4. removing (remove(index))
5. removing all(clear())
6. finding(available or not) (contains(Object))
7. getting the data (get(-))
8. checking empty or not (empty())
9. replacing elements (set(-,-))
10. removing common/duplicate elements(removeAll())
11. removing unique/individual elements(retainAll())
12. adding the elements middle of collection object

Collection: is a group of individual,homogeneous,heterogeneous,duplicate objects with in a single entity is called collection.



Drawback Of Object array:

1. Size is fixed.
2. Not avoiding duplicate data.
3. Not providing any sorting order.
4. Not providing any searching technique. not
5. Without replacing old data with new data, we can't place the data into array.
6. While placing the data or deleting data from array I need to shift elements from left to right or right to left, that flexibility not available in arrays.
7. Forced to array for reading the data in sequence order.

Collection framework:

- ✓ Collection framework is a class library to handle group of objects.
- ✓ It is implemented in java.util package. It has been included in java 2.0.

- ✓ It is collection of classes and interface.
- ✓ Each and every class and interface having its own priority and advantages.

Advantages of Collection:

- The size must be increase dynamically, based on our application requirement.
 - We can overcome the problem of memory wastage.
 - We will get the application efficiency and performance.
 - The size will be not only increasing but also decreasing; we can see this nature in ArrayList in briefly.
- We collect/store different type and same type of objects in to one variable. That means we can store homogeneous and heterogeneous objects in our collection variable.
- Collection having readymade/predefine methods for manipulate the objects.
- Based on the requirement, we can store the duplication code and avoiding the duplication code or data redundancy.
- All Collection classes are implements Cloneable, Serializable interfaces.
- Both ArrayList and Vector classes implements RandomAccess interface also.
- All Collection class override the `toString()` of Object class.

Collection:

- Collection is a root interface for representing a group of objects nothing but elements as a single entity.
- Collection doesn't allow duplicates and some are allow duplicate values.
- Collection are having order and some not having any order.
- There is no direct implementation of this interface.

Collections:

Collections is an utility class available in `java.util` package for defining several utility methods for collection objects.

Types of Collection Framework:

Based on the way of storing the objects, the collection framework is categorized into two approaches.

- (a) Collection hierarchy.
- (b) Map hierarchy.

Collection hierarchy:

Mainly it has been classified into three categories.

- (a) List
- (b) Set
- (c) Queues

Map hierarchy:

In the map hierarchy in the objects will be stored in the order of key-value pairs.

Collection interface:

- Collection is a root interface.
- It represents group of objects into single entity.
- It is having common methods, which can be applied on any objects.

Methods in collection interface:

```
E:\>javap java.util.Collection
Compiled from "Collection.java"
public interface java.util.Collection extends java.lang.Iterable{
    public abstract int size();
    public abstract boolean isEmpty();
    public abstract boolean contains(java.lang.Object);
    public abstract java.util.Iterator iterator();
    public abstract java.lang.Object[] toArray();
    public abstract java.lang.Object[] toArray(java.lang.Object[]);
    public abstract boolean add(java.lang.Object);
    public abstract boolean remove(java.lang.Object);
    public abstract boolean containsAll(java.util.Collection);
    public abstract boolean addAll(java.util.Collection);
    public abstract boolean removeAll(java.util.Collection);
    public abstract boolean retainAll(java.util.Collection);
    public abstract void clear();
    public abstract boolean equals(java.lang.Object);
    public abstract int hashCode();
```

List interface:

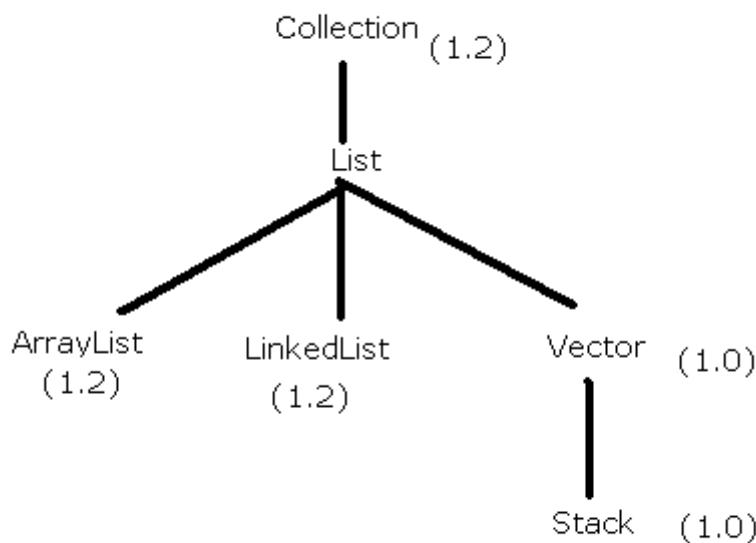
- List is a child interface of Collection.

- It is used for group of individual objects as single entity.
- It can allow the duplicate values, null values and zeros (more than one time).
- We can differentiate duplicate values by using index.
- It is having an order of insertion by using index.
- It has been included in java 1.2.
- We can also call as sequence.

```

Iterable                                -->allow Homogeneous
|                                         -->allow Hetrogeneous
Collection                            -->Dynamically growable in nature
|                                         -->Order is preserved
|                                         -->toString() override
|                                         -->implements Serializable,Cloneable
|                                         -->AL,V-->implements RandomAccess
|                                         -->AL,V,S-->default size(capacity) is 10
|                                         -->null allows
|                                         -->more than one null allow
|                                         -->duplicate values allows
|                                         -->No sorting order by default
|                                         --> Searching technique by default
|                                         -->AL loadfactor(fill ratio)--(cc*3/2)+1;
|                                         -->V,S(loadfactor---cc*2)
|                                         -->Stack

```



Methods in List Interface:

```
F:\>javap java.util.List
Compiled from "List.java"
public interface java.util.List extends java.util.Collection{
    public abstract int size();
    public abstract boolean isEmpty();
    public abstract boolean contains(java.lang.Object);
    public abstract java.util.Iterator iterator();
    public abstract java.lang.Object[] toArray();
    public abstract java.lang.Object[] toArray(java.lang.Object[]);
    public abstract boolean add(java.lang.Object);
    public abstract boolean remove(java.lang.Object);
    public abstract boolean containsAll(java.util.Collection);
    public abstract boolean addAll(java.util.Collection);
    public abstract boolean addAll(int, java.util.Collection);
    public abstract boolean removeAll(java.util.Collection);
    public abstract boolean retainAll(java.util.Collection);
    public abstract void clear();
    public abstract boolean equals(java.lang.Object);
    public abstract int hashCode();
    public abstract java.lang.Object get(int);
    public abstract java.lang.Object set(int, java.lang.Object);
    public abstract void add(int, java.lang.Object);
    public abstract java.lang.Object remove(int);
    public abstract int indexOf(java.lang.Object);
    public abstract int lastIndexOf(java.lang.Object);
    public abstract java.util.ListIterator listIterator();
    public abstract java.util.ListIterator listIterator(int);
    public abstract java.util.List subList(int, int);
}
```

ArrayList Class:

- It is the child class of List interface.
- ArrayList is a growable and resizable array.
- It allows the duplicate values.
- Heterogeneous (different) objects are allowed.
- Null insertion is possible.
- Insertion order is preserved.
- It is override the `toString()` method.

Constructors:

- (1) `ArrayList l=new ArrayList();`

Here capacity is empty/null.

The default initial capacity is ten (10).

- (2) If the ArrayList reaches its initial capacity, the new ArrayList will be created, with the $(\text{current capacity} * 3/2) + 1$.

Example:

The minimum capacity is 10. Then after the new capacity is
 $(10*3/2)+1=16$.

- (3) ArrayList l=new ArrayList(int initial capacity);
- (4) ArrayList l=new ArrayList(Collection c)

Here ArrayList will create with the equality size of collection object.

Used at inter conversion between collection objects.

Methods in ArrayList:

```
F:\>javap java.util.ArrayList
Compiled from "ArrayList.java"
public class java.util.ArrayList extends java.util.AbstractList implements java.util.List,java.util.RandomAccess,java.lang.Cloneable,java.io.Serializable{
    public java.util.ArrayList(int);
    public java.util.ArrayList();
    public java.util.ArrayList(java.util.Collection);
    public void trimToSize();
    public void ensureCapacity(int);
    public int size();
    public boolean isEmpty();
    public boolean contains(java.lang.Object);
    public int indexOf(java.lang.Object);
    public int lastIndexOf(java.lang.Object);
    public java.lang.Object clone();
    public java.lang.Object[] toArray();
    public java.lang.Object[] toArray(java.lang.Object[]);
    public java.lang.Object get(int);
    public java.lang.Object set(int, java.lang.Object);
    public boolean add(java.lang.Object);
    public void add(int, java.lang.Object);
    public java.lang.Object remove(int);
    public boolean remove(java.lang.Object);
    public void clear();
    public boolean addAll(java.util.Collection);
```

- ArrayList is implements Serializable and Clonable interfaces.
- ArrayList implements the RandomAccess interface.
- Then all the elements in the ArrayList can be retrieved with same speed.
- That's why ArrayList is best suitable for "fast retrieval".
- The drawback of the ArrayList is not suitable for frequent insertion and deletion operations at middle of ArrayList.

- The reason is if we insert or delete the elements at middle of the array the shift operations will be happened, that why it will take more time. This is same for Vector class also.

Program on ArrayList:

```
import java.util.*;
class ArrayListDemo {
    public static void main(String args[]) throws Exception{
        System.out.println("good and bad morning");
        ArrayList l=new ArrayList();
        l.add("r");
        l.add("a");
        l.add("m");
        l.add("u");
        l.add("s");
        l.remove(4);
        System.out.println(l);
    }
}
```

Output:

```
c:\ C:\WINDOWS\system32\cmd.exe
good ***** morning
[r
,a
,m
,u
]
Press any key to continue . . .
```

```
import java.util.*;
```

```
class ArrayListDemo
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
        ArrayList a=new ArrayList();
        a.add("A");
        a.add(".");
        a.add("ra");
        a.add("m");
        a.add(0,"F");
        a.add("A");
        a.add(null);
        System.out.println(a);
        System.out.println(a.size());
        System.out.println(a.isEmpty());
        System.out.println(a.contains("."));
        System.out.println(a.indexOf("ra"));
        System.out.println(a.lastIndexOf("ra"));
        Object o=a.clone();
        System.out.println(o);
        Object o1[]=a.toArray();
        System.out.println(o1[0]);
        System.out.println(o1[2]);
```

```
        System.out.println(o1[4]);
        System.out.println(o1[3]);
//      a.clear();
        System.out.println(a);
        System.out.println(a.get(5));
        System.out.println(a.remove(5));
        System.out.println(a);
        a.ensureCapacity(4);
        System.out.println(a.size());
        a.trimToSize();
        System.out.println(a.size());
    }
}

//the default size is 10. and load factor is 3/2+1;
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Hello World!
[F, A, , ra, m, A, null]
7
false
true
3
3
[F, A, , ra, m, A, null]
F
.
m
ra
[F, A, , ra, m, A, null]
A
A
[F, A, , ra, m, A, null]
6
6
Press any key to continue . . .
```

ArrayList is not suitable for frequent insertion and deletion at middle of ArrayList, and then to overcome this problem we have a new class called "LinkedList".

LinkedList:

- It is the child class of List interface, Deque, Queue.
- LinkedList class extends AbstractSequentialList .
- Duplicate objects will be allowed.
- Null insertion can be allowed.
- Insertion order will be preserved.
- Heterogeneous object can be allowed.
- It is the best suitable for frequent insertion and deletion operations at middle of the list.
- Compare to ArrayList, the LinkedList will not be suitable for only frequent retrieval operation, why because it not implement the RandomAccess interface.
- It is also implement the Serializable and Clonable interfaces.
- Up to java1.4 the LinkedList only implements the List interface, whereas in java1.5 LinkedList is also implements the Queue interface.

Constructors in LinkedList:

```
LinkedList l=new LinkedList();
```

```
LinkedList l=new LinkedList(Collection c);
```

Methods in LinkedList:

```
E:\>javap java.util.LinkedList
Compiled from "LinkedList.java"
public class java.util.LinkedList extends java.util.AbstractSequentialList implements java.util.List,java.util.Deque,java.lang.Cloneable,java.io.Serializable{
    public java.util.LinkedList();
    public java.util.LinkedList(java.util.Collection);
    public java.lang.Object getFirst();
    public java.lang.Object getLast();
    public java.lang.Object removeFirst();
    public java.lang.Object removeLast();
    public void addFirst(java.lang.Object);
    public void addLast(java.lang.Object);
    public boolean contains(java.lang.Object);
    public int size();
    public boolean add(java.lang.Object);
    public boolean remove(java.lang.Object);
    public boolean addAll(java.util.Collection);
    public boolean addAll(int, java.util.Collection);
    public void clear();
    public java.lang.Object get(int);
    public java.lang.Object set(int, java.lang.Object);
    public void add(int, java.lang.Object);
    public java.lang.Object remove(int);
    public int indexOf(java.lang.Object);
    public int lastIndexOf(java.lang.Object);
    public java.lang.Object peek();
    public java.lang.Object element();
    public java.lang.Object poll();
    public java.lang.Object remove();
    public boolean offer(java.lang.Object);
    public boolean offerFirst(java.lang.Object);
    public boolean offerLast(java.lang.Object);
    public java.lang.Object peekFirst();
    public java.lang.Object peekLast();
    public java.lang.Object pollFirst();
    public java.lang.Object pollLast();
    public void push(java.lang.Object);
    public java.lang.Object pop();
    public boolean removeFirstOccurrence(java.lang.Object);
    public boolean removeLastOccurrence(java.lang.Object);
    public java.util.ListIterator listIterator(int);
    public java.util.Iterator descendingIterator();
    public java.lang.Object clone();
    public java.lang.Object[] toArray();
    public java.lang.Object[] toArray(java.lang.Object[]);
    static java.util.LinkedList$Entry access$000(java.util.LinkedList);
    static int access$100(java.util.LinkedList);
    static java.lang.Object access$200(java.util.LinkedList, java.util.LinkedList$Entry);
    static java.util.LinkedList$Entry access$300(java.util.LinkedList, java.lang.Object, java.util.LinkedList$Entry);
}
```

```
import java.util.*;
```

```
class LinkedListDemo
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        System.out.println("Hello World!");
```

```
ArrayList al=new ArrayList();
al.add("kiran");
al.add("suji");
al.add("vani");
System.out.println(al);
LinkedList ll=new LinkedList();
ll.add("a");
ll.add(10);
ll.add(new Integer(100));
ll.add(null);
ll.add(2,"ramu");
System.out.println(ll);
ll.add(1,al);
System.out.println(ll);
ll.add(al);
System.out.println(ll);
ll.addFirst("first");
ll.addLast("last");
System.out.println(ll);
Object o=ll.clone();
System.out.println("clone values are:"+o);
System.out.println(ll.contains("ramu"));
System.out.println(ll.getFirst());
System.out.println(ll.getLast());
```

```
System.out.println(l1.get(5));
System.out.println(l1.get(2));
System.out.println(l1.indexOf("ramu"));
System.out.println(l1.lastIndexOf("first"));
System.out.println(l1.remove(5));
System.out.println(l1.removeFirst());
System.out.println(l1.removeLast());
l1.set(0,100);
System.out.println(l1);
Object o1[]=l1.toArray();
System.out.println(o1[1]);
}
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
Hello World!
[kiran, suji, vani]
[a, 10, ramu, 100, null]
[a, [kiran, suji, vani], 10, ramu, 100, null]
[a, [kiran, suji, vani], 10, ramu, 100, null, [kiran, suji, vani]]
[first, a, [kiran, suji, vani], 10, ramu, 100, null, [kiran, suji, vani], last]
clone values are:[first, a, [kiran, suji, vani], 10, ramu, 100, null, [kiran, suji, vani], last]
true
first
last
100
[kiran, suji, vani]
4
0
100
first
last
[100, [kiran, suji, vani], 10, ramu, null, [kiran, suji, vani]]
[kiran, suji, vani]
Press any key to continue . . .
```

Vector class:

- It is also child class of List interface.
- It allows the duplicate values.
- It is also allows the null values.
- The order must be preserved.
- It has been available from java1.0 onwards.
- Like ArrayList, it is also best suitable for frequent retrieval operations, the reason it can be implements RandomAccess interface.
- Like ArrayList, it is also not suitable for frequent insertion and deletion operations in the middle.
- It is also implements the Clonable, Serializable interface.
- It is somewhat similar to ArrayList but there are some difference between ArrayList and Vector.
- Methods in ArrayList are not synchronized, whereas Vector methods are synchronized.
- ArrayList is not thread safe, whereas Vector is a thread safe.
- Performance is high in ArrayList, where as low in Vector.
- ArrayList by default non-synchronized, where as Vector is synchronized.
- We can get synchronized ArrayList by using following operation. That is
 - Public static List synchronizedList(ArrayList l);

Example:

```
ArrayList      l=new ArrayList();
List          l1= Collections.synchronizedList(l);
```

Methods in Vector:

```
F:\\>javap java.util.Vector
Compiled from "Vector.java"
public class java.util.Vector extends java.util.AbstractList implements java.util.List,java.util.RandomAccess,java.lang.Cloneable,java.io.Serializable{
    protected java.lang.Object[] elementData;
    protected int elementCount;
    protected int capacityIncrement;
    public java.util.Vector(int, int);
    public java.util.Vector();
    public java.util.Vector<>;
    public java.util.Vector(java.util.Collection);
    public synchronized void copyInto(java.lang.Object[]);
    public synchronized void trimToSize();
    public synchronized void ensureCapacity(int);
    public synchronized void setSize(int);
    public synchronized int capacity();
    public synchronized int size();
    public synchronized boolean isEmpty();
    public java.util.Enumeration elements();
    public boolean contains(java.lang.Object);
    public int indexOf(java.lang.Object);
    public synchronized int indexOf(java.lang.Object, int);
    public synchronized int lastIndexOf(java.lang.Object);
    public synchronized int lastIndexOf(java.lang.Object, int);
    public synchronized java.lang.Object elementAt(int);
    public synchronized java.lang.Object firstElement();
    public synchronized java.lang.Object lastElement();
    public synchronized void setElementAt(java.lang.Object, int);
    public synchronized void removeElementAt(int);
    public synchronized void insertElementAt(java.lang.Object, int);
    public synchronized void addElement(java.lang.Object);
    public synchronized boolean removeElement(java.lang.Object);
    public synchronized void removeAllElements();
    public synchronized java.lang.Object clone();
    public synchronized java.lang.Object[] toArray();
    public synchronized java.lang.Object[] toArray(java.lang.Object[]);
    public synchronized java.lang.Object get(int);
    public synchronized java.lang.Object set(int, java.lang.Object);
    public synchronized boolean add(java.lang.Object);
    public boolean remove(java.lang.Object);
    public void add(int, java.lang.Object);
    public synchronized java.lang.Object remove(int);
    public void clear();
    public synchronized boolean containsAll(java.util.Collection);
    public synchronized boolean addAll(java.util.Collection);
    public synchronized boolean removeAll(java.util.Collection);
    public synchronized boolean retainAll(java.util.Collection);
    public synchronized boolean addAll(int, java.util.Collection);
    public synchronized boolean equals(java.lang.Object);
    public synchronized int hashCode();
    public synchronized java.lang.String toString();
    public synchronized java.util.List subList(int, int);
    protected synchronized void removeRange(int, int);
```

Constructors in Vector:

(1) Vector v =new Vector()

Initial capacity is 10.

(2) Vector v=new Vector (int x);

The vector class size will be doubled after reached its max capacity.

(3)Vector v=new Vector (int initial capacity, int increment capacity);

(4)Vector v=new Vector (Collection c)

Stack:

It is the child class of vector.

```
c:\ C:\WINDOWS\system32\cmd.exe
::>javap java.util.Stack
Compiled from "Stack.java"
public class java.util.Stack extends java.util.Vector{
    public java.util.Stack();
    public java.lang.Object push(java.lang.Object);
    public synchronized java.lang.Object pop();
    public synchronized java.lang.Object peek();
    public boolean empty();
    public synchronized int search(java.lang.Object);
}
```

Retrieving elements from Collections:

There are four ways to retrieve the element/objects from Collections objects.

They are:

- (a) For-each method.
- (b) Iterator.
- (c) ListIterator.
- (d) Enumeration.

The above b, c, d are all interfaces.

For-each loop:

It is a loop like for loop, which executes group of statements for each element of the collection.

Example:

```
For (variable: collection-object)
{
    Statements;
}
```

Here collection-object having how many elements, that much of size will be assign into variable and that much of times statements will be executed.

Enumeration:

Enumeration is an interface, which is introduced in java 1.0.

It is useful for retrieving the elements from collection object.

Methods in Enumeration:

```
java>javap -javacutil Enumeration
Compiled from "Enumeration.java"
public interface java.util Enumeration{
    public abstract boolean hasMoreElements();
    public abstract java.lang.Object nextElement();
}
```

By using elements () method, we can get Enumeration object.

Elements() method will be available in Vector class in java.util package.

DrawBacks:

- This Enumeration interface doesn't have any remove method. It has only read only methods.
- It is only applicable for Legacy class.
- It is a single direction cursor.

Legacy class:

- Legacy classes are those that were built using java 1.1 and java 1.2.
- In general these classes are called as java classes.
- In java 1.0, Vector class was available instead of dynamic array and since there were no ArrayList class people were forced to use Vector for this purpose.
- When java 1.2 was released ArrayList was much more advanced to Vector but has all the features of Vector too.
- So people started using ArrayList instead of Vector.
- And in this case Vector became legacy class.
- Legacy classes are introduced in 1.0 and re-engineered into java 1.2 versions.

Note:

But in general a “legacy” is a term used to define software created using older version of software.

Iterator:

- Iterator is an interface, which was introduced in java 1.2.
- It contains methods to retrieve the elements one by one from a collection object.
- It has 3 methods.

```
F:\>javap -java.util.Iterator
Compiled from "Iterator.java"
public interface java.util.Iterator{
    public abstract boolean hasNext();
    public abstract java.lang.Object next();
    public abstract void remove();
}
```

- It will work on any type of classes that is any Collection implemented class.
- By using iterator () method of ArrayList also we can get Iterator object.
- This method will available in List interface.

Drawbacks:

It is also single direction cursor.

It does not have operation like add and replace of new objects.

ListIterator:

- ListIterator is an interface that contains methods to retrieve the elements one by one from a collection object, both in forward and reverse direction.
- It has 9 methods.

```
F:\>javap -java.util.ListIterator
Compiled from "ListIterator.java"
public interface java.util.ListIterator extends java.util.Iterator{
    public abstract boolean hasNext();
    public abstract java.lang.Object next();
    public abstract boolean hasPrevious();
    public abstract java.lang.Object previous();
    public abstract int nextIndex();
    public abstract int previousIndex();
    public abstract void remove();
    public abstract void set(java.lang.Object);
    public abstract void add(java.lang.Object);
}
```

- It is the child interface of Iterator interface.

- It is the bi-directional cursor.
- It has additional feature when compare Iterator and enumeration, that is used for read, remove, replace, add the object.

Enumeration	Iterator	ListIterator
--> 1.0 --> 2 methods --> 1.we can check whether the elements are existed or not 2. if existed read the elements ---> Legacy classes single direction	1.2 4 methods --> 1.we can check whether the elements are existed or not 2. if existed read the elements 3.remove the data --->For all Collection objects Single direction	1.2 9 methods --> 1.we can check whether the elements are existed or not 2. if existed read the elements 3.remove the data 4. add the data 5. replace the data --->list implement s classes double direction

Set Interface:

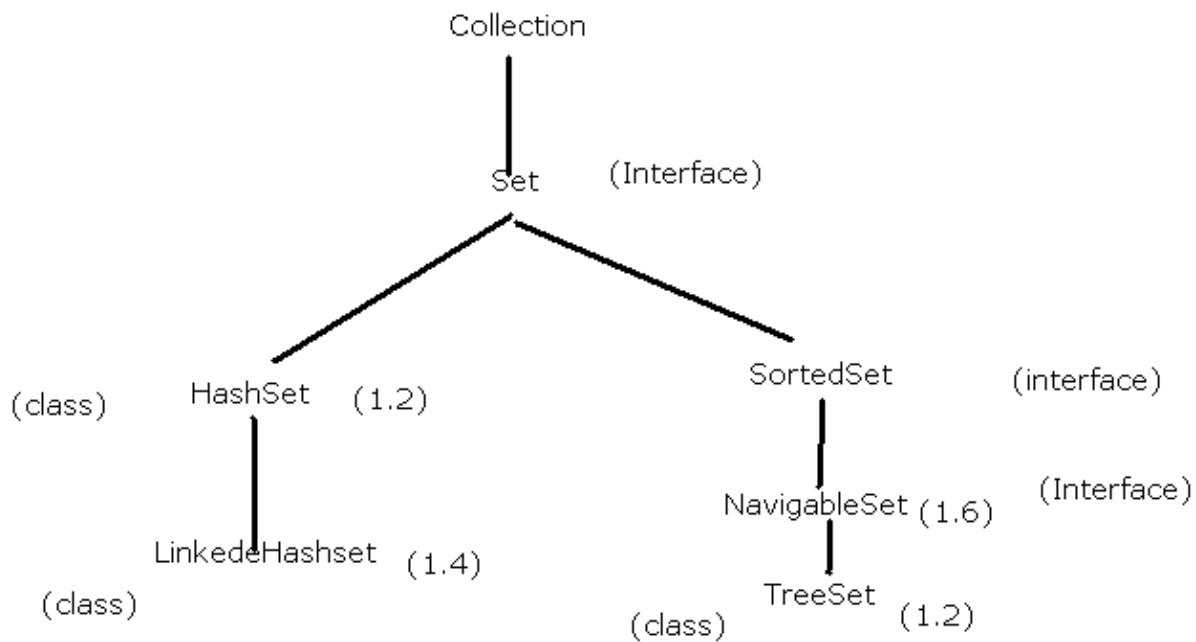
- A Set interface represents a set of elements.
- It doesn't allow the duplicate elements.
- There is no proper preserve of insertion.

Methods in Set:

```
%javap java.util.Set
Compiled from "Set.java"
public interface java.util.Set extends java.util.Collection{
    public abstract int size();
    public abstract boolean isEmpty();
    public abstract boolean contains(java.lang.Object);
    public abstract java.util.Iterator iterator();
    public abstract java.lang.Object[] toArray();
    public abstract java.lang.Object[] toArray(java.lang.Object[]);
    public abstract boolean add(java.lang.Object);
    public abstract boolean remove(java.lang.Object);
    public abstract boolean containsAll(java.util.Collection);
    public abstract boolean addAll(java.util.Collection);
    public abstract boolean retainAll(java.util.Collection);
    public abstract boolean removeAll(java.util.Collection);
    public abstract void clear();
    public abstract boolean equals(java.lang.Object);
    public abstract int hashCode();
}
```

These methods same as Collection interface methods.

Set interface doesn't contain any extra methods from Collection.



	<i>List:</i>	<i>HashSet</i>	<i>LinkedHashSet</i>	<i>TreeSet</i>
1) Homogeneous	allows	allows	allows	allows
2) Hetrogeneous	"	"	"	no
3) duplicate data	"	no	no	no
4) Orderd is preserved	"	no	Yes	no
5) null values	"	allow	allows	no
5)morethan one null value	"	no	no	no
6)sorting order	no	no	no	yes
7)override <i>toString</i>	yes	yes	yes	yes
8)Serializable Cloneable	Yes	yes	yes	yes

HashSet:

- HashSet is a class, which represents set of elements (objects).
- Insertion order is not preserved.
- Duplication objects will not allowed.
- Heterogeneous objects will be allowed.
- Like List null insertion is possible, but only once.
- It is based on hashCode of an object.

- It is the best suitable for searching operation.
- It implements Serializable and Clonable interfaces.
- Its underlying data structure is HashTable.
- The default inserting order is, if we insert first element there is no specific rule followed by JVM. After that inserting the second element, here JVM will compare the hashCode of first element to hashCode of second element.
- If those two objects hashCode equal then, JVM will use the equals method on those two objects, if the return type true then, the JVM will decide to these two objects are duplicates, it place only one element in the HashSet.
- If equals () method return false then JVM will place these two objects into HashSet.
- If hashCode of the two objects are not equal in the first case, these two objects will place in HashSet.

Methods in HashSet:

```
F:\>javap java.util.HashSet
Compiled from "HashSet.java"
public class java.util.HashSet extends java.util.AbstractSet implements java.util.Set,java.lang.Cloneable,java.io.Serializable{
    static final long serialVersionUID;
    public java.util.HashSet();
    public java.util.HashSet(java.util.Collection);
    public java.util.HashSet(int, float);
    public java.util.HashSet(int);
    java.util.HashSet(int, float, boolean);
    public java.util.Iterator iterator();
    public int size();
    public boolean isEmpty();
    public boolean contains(java.lang.Object);
    public boolean add(java.lang.Object);
    public boolean remove(java.lang.Object);
    public void clear();
    public java.lang.Object clone();
    static {};
}
```

Constructors in HashSet:

(1) HashSet hs=new HashSet();

The default capacity is 16.

Load Factor is 0.75.

(2) HashSet hs=new HashSet(int capacity)

(3) HashSet hs=new HashSet(int initial capacity, float load factor);

Load factor must be in between 0 to 1.

(4) HashSet hs=new HashSet(collection c);

```
public class CollectionDemo {
    public static void main(String[] args) {
        Integer i = new Integer(10);
        Integer j = new Integer(20);
        Integer k = new Integer(20);
        HashSet hs = new HashSet();
        hs.add(i);
        hs.add(j);
        hs.add(k);
        System.out.println(hs);

        HashSet hs1 = new HashSet();
        hs1.add("Aa");
        hs1.add("BB");
        System.out.println(hs1);
        System.out.println("Aa".hashCode());
        System.out.println("BB".hashCode());

    }
}

import java.util.Iterator;
import java.util.Scanner;
import java.util.concurrent.CopyOnWriteArraySet;

public class CollectionDemo {
    public static void main(String[] args) {
        /*HashSet hs = new HashSet();
        hs.add(20);
        hs.add(30);
        hs.add(10);
        hs.add(40);
        System.out.println(hs);

        Iterator i = hs.iterator();
        while(i.hasNext()){
            System.out.println(i.next());
            hs.add("ram");
        }*/
        CopyOnWriteArraySet cas = new CopyOnWriteArraySet();
        cas.add(20);
        cas.add(30);
        cas.add(10);
        cas.add(40);
        System.out.println(cas);
    }
}
```

```

Iterator i = cas.iterator();
while(i.hasNext()){
    System.out.println(i.next());
    //cas.add("ram");
    cas.add(new Scanner(System.in).next());
}
System.out.println(cas);
}
}

```

LinkedHashSet:

This is the sub class of HashSet.

It doesn't have any additional methods.

```

D:\Java\java.util.LinkedHashSet
Compiled from "LinkedHashSet.java"
public class java.util.LinkedHashSet extends java.util.HashSet implements java.util.Set,java.lang.Cloneable,java.io.Serializable{
    public java.util.LinkedHashSet(int, float);
    public java.util.LinkedHashSet();
    public java.util.LinkedHashSet();
    public java.util.LinkedHashSet(java.util.Collection);
}

```

Features:

Introduced in java 1.4.

Underlying data structure is LinkedList + HashTable. That's why it maintains the insertion order.

It doesn't allow the duplicate values that are why it is best suited for cache memory application.

SortedSet:

SortedSet is a child interface of Set interface.

Its name specifies that, it will be useful for grouping the unique object according to some sorting order.

The sorting order may be default or customized sorting order.

Methods in SortedSet:

```
y:\>javap -javac -java.util.SortedSet  
Compiled from "SortedSet.java"  
public interface java.util.SortedSet extends java.util.Set{  
    public abstract java.util.Comparator comparator();  
    public abstract java.util.SortedSet subSet(java.lang.Object, java.lang.Object);  
    public abstract java.util.SortedSet headSet(java.lang.Object);  
    public abstract java.util.SortedSet tailSet(java.lang.Object);  
    public abstract java.lang.Object first();  
    public abstract java.lang.Object last();  
}
```

comparator():

This method will return Comparator interface.

If the sorting technique is default, then this method returns null.

TreeSet:

- The underlying data structure is balanced tree.
- Duplicate values are not allowed.
- Insertion order is preserved.
- Heterogeneous values are not allowed.

Constructors in TreeSet:

(1) TreeSet ts=new TreeSet().

Creates an empty TreeSet object, where the sorting order is default natural sorting order.

The elements which are inserted into the set must implement Comparable interface.

(2) TreeSet ts=new TreeSet(Comparator obj)

Creates empty TreeSet object, where sorting order is specific by Comparator.

This is customized sorting order.

(3) TreeSet ts=new TreeSet(Collection c)

(4) TreeSet ts=new TreeSet(SortedSet s)

- Creates a new tree set containing the same elements and using the same ordering as the specified SortedSet.

- For empty TreeSet as the first value null insertion is possible, but after inserting null value, if we are trying insert any other null value then we will get NullPointerException.
- Whereas non-empty, if we are inserting null value, then we will get NullPointerException.
- If we want depend on natural sorting order, then we should place the homogeneous objects and comparable otherwise we will get ClassCastException.
- If an object is a comparable, if and only if the corresponding class will implements Comparable interface.
- String and all wrapper classes implemented Comparable interface.
- Whereas StringBuffer, StringBuilder does not implement the Comparable interface.

```

E:\practice>javap java.util.TreeSet
Compiled from "TreeSet.java"
public class java.util.TreeSet extends java.util.AbstractSet implements java.util.NavigableSet,java.lang.Cloneable,java.io.Serializable{
    java.util.TreeSet(java.util.Map);
    public java.util.TreeSet();
    public java.util.TreeSet(java.util.Comparator);
    public java.util.TreeSet(java.util.Collection);
    public java.util.TreeSet(java.util.SortedSet);
    public java.util.Iterator iterator();
    public java.util.Iterator descendingIterator();
    public java.util.NavigableSet descendingSet();
    public int size();
    public boolean isEmpty();
    public boolean contains(java.lang.Object);
    public boolean add(java.lang.Object);
    public boolean remove(java.lang.Object);
    public void clear();
    public boolean addAll(java.util.Collection);
    public java.util.NavigableSet subSet(java.lang.Object, boolean, java.lang.Object, boolean);
    public java.util.NavigableSet headSet(java.lang.Object, boolean);
    public java.util.NavigableSet tailSet(java.lang.Object, boolean);
    public java.util.SortedSet subSet(java.lang.Object, java.lang.Object);
    public java.util.SortedSet headSet(java.lang.Object);
    public java.util.SortedSet tailSet(java.lang.Object);
    public java.util.Comparator comparator();
    public java.lang.Object first();
    public java.lang.Object last();
    public java.lang.Object lower(java.lang.Object);
    public java.lang.Object floor(java.lang.Object);
    public java.lang.Object ceiling(java.lang.Object);
    public java.lang.Object higher(java.lang.Object);
    public java.lang.Object pollFirst();
    public java.lang.Object pollLast();
    public java.lang.Object close();
}

```

NavigableSet:

For navigation of SortedSet purpose, the sun micro people introduce a new concept called NavigableSet.

It is the child interface of SortedSet interface.

It has some method for navigation purpose.

Methods in NavigableSet:

```
>>>javap -java.util.NavigableSet
Compiled from "NavigableSet.java"
public interface java.util.NavigableSet extends java.util.SortedSet<
    public abstract java.lang.Object lower(java.lang.Object);
    public abstract java.lang.Object floor(java.lang.Object);
    public abstract java.lang.Object ceiling(java.lang.Object);
    public abstract java.lang.Object higher(java.lang.Object);
    public abstract java.lang.Object pollFirst();
    public abstract java.lang.Object pollLast();
    public abstract java.util.Iterator iterator();
    public abstract java.util.NavigableSet descendingSet();
    public abstract java.util.Iterator descendingIterator();
    public abstract java.util.NavigableSet subSet(java.lang.Object, boolean, java.lang.Object, boolean);
    public abstract java.util.NavigableSet headSet(java.lang.Object, boolean);
    public abstract java.util.NavigableSet tailSet(java.lang.Object, boolean);
    public abstract java.util.SortedSet subSet(java.lang.Object, java.lang.Object);
>>>    public abstract java.util.SortedSet headSet(java.lang.Object);
    public abstract java.util.SortedSet tailSet(java.lang.Object);
```

Comparable Interface:

It is available in java.lang package and contains only one method. That is

```
Public abstract int compareTo(java.lang.Object obj);
```

```
Obj1.compareTo(obj2)
```

Rules:

- (1) Return positive number if obj1 comes before obj2.
- (2) Return negative number if obj1 comes after obj2.
- (3) Return zero, if obj1 and obj2 are equal.

When we inserted elements into TreeSet, if we are not using any sorting order, then we should call compareTo() method and default natural sorting order.

Comparator interface:

If we want inserted elements into TreeSet with our own or customized sorting technique then we should go for Comparator interface.

This interface will be available in java.util package.

Methods in Comparator:

It has two methods.

```
javap -java.util.Comparator
Compiled from "Comparator.java"
public interface java.util.Comparator<
    public abstract int compare(java.lang.Object, java.lang.Object);
    public abstract boolean equals(java.lang.Object);
>
```

(1)Public abstract int compare(obj1,obj2):

Rules:

- (1) Return positive number if obj1 comes before obj2.
- (2) Return negative number if obj1 comes after obj2.
- (3) Return zero, if obj1 and obj2 are equal.

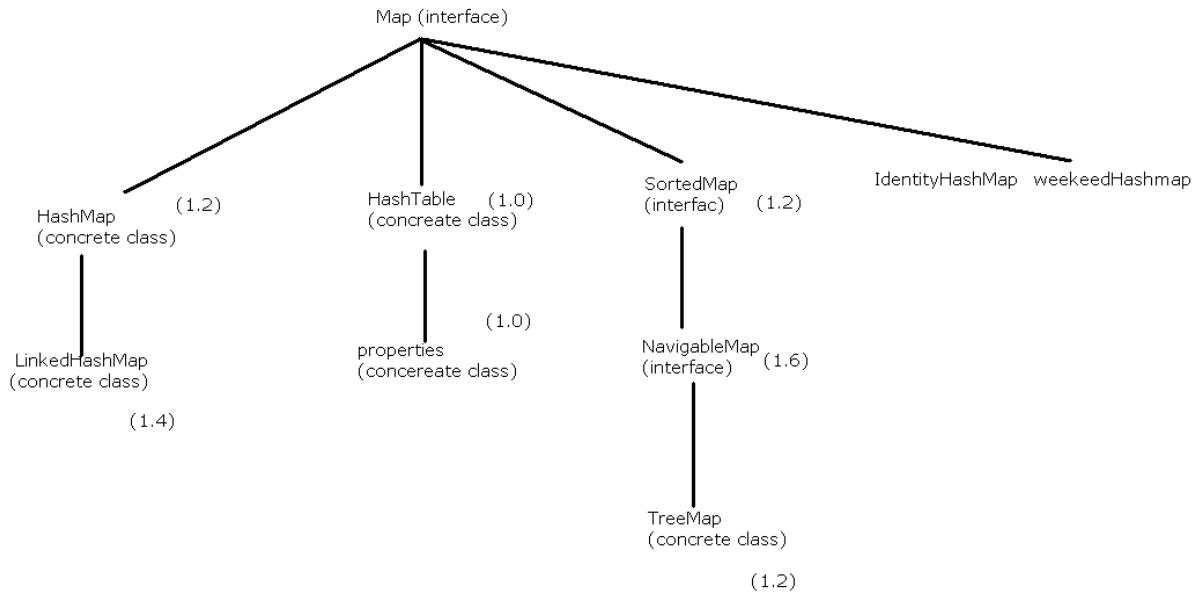
(2)public abstract boolean equals(Obj);

- If we using Comparator interface, we would provide implementation of compare(obj1,obj2).
- Implementing equals() is optional, because it has implementation in Object class.
- We can provide heterogeneous objects also.

Map Hierarchy:

Map hierarchy classes are used to collect elements in (key, value) pair of format.

Both key and value are objects only.



Map Interface:

- In a Map interface key should be unique, whereas value can be duplicate.
- The pair of (key, value) is called one entry.
- Map interface is not a child interface of Collection interface.

Methods in Map:

```

>>>javap java.util.Map
Compiled from "Map.java"
public interface java.util.Map{
    public abstract int size();
    public abstract boolean isEmpty();
    public abstract boolean containsKey(java.lang.Object);
    public abstract boolean containsValue(java.lang.Object);
    public abstract java.lang.Object get(java.lang.Object);
    public abstract java.lang.Object put(java.lang.Object, java.lang.Object);
    public abstract java.lang.Object remove(java.lang.Object);
    public abstract void putAll(java.util.Map);
    public abstract void clear();
    public abstract java.util.Set keySet();
    public abstract java.util.Collection values();
    public abstract java.util.Set entrySet();
    public abstract boolean equals(java.lang.Object);
    public abstract int hashCode();
}
  
```

Entry Interface:

A map is a collection of entries (key, value), hence Entry is an inner interface defined in Map interface.

It has 5 methods.

```
javap -c java.util.Map$Entry
Compiled from "Map.java"
public interface java.util.Map$Entry{
    public abstract java.lang.Object getKey();
    public abstract java.lang.Object getValue();
    public abstract java.lang.Object setValue(java.lang.Object);
    public abstract boolean equals(java.lang.Object);
    public abstract int hashCode();
}
```

HashMap class:

- HashMap is a collection that stores elements in the form of key-value pairs.
- If key is provided later, its corresponding value can be easily retrieved from the HashMap.
- Keys should be unique.
- Value may or may be duplicate.
- The underlying data structure is Hashtable.
- It is not synchronized, if multiple threads work on this object, we might get unreliable values.
- Insertion order is not preserved.
- Heterogeneous object can be allowed.
- It has been based on hashCode of the keys.
- Null insertion can be possible for both key and values.
- It has been introduced in java 1.2.
- HashMap by default it is not a synchronized, but make it as synchronized(thread safe).
- By using Collection.synchronizedMap(java.util.Map object) we make it as synchronized.

Methods in HashMap:

```
F:\>javap java.util.HashMap
Compiled from "HashMap.java"
public class java.util.HashMap extends java.util.AbstractMap implements java.util.Map,java.lang.Cloneable,java.io.Serializable{
    static final int DEFAULT_INITIAL_CAPACITY;
    static final int MAXIMUM_CAPACITY;
    static final float DEFAULT_LOAD_FACTOR;
    transient java.util.HashMap$Entry[] table;
    transient int size;
    int threshold;
    final float loadFactor;
    volatile transient int modCount;
    public java.util.HashMap(int, float);
    public java.util.HashMap(int);
    public java.util.HashMap();
    public java.util.HashMap(java.util.Map);
    void init();
    static int hash(int);
    static int indexFor(int, int);
    public int size();
    public boolean isEmpty();
    public java.lang.Object get(java.lang.Object);
    public boolean containsKey(java.lang.Object);
    final java.util.HashMap$Entry getEntry(java.lang.Object);
    public java.lang.Object put(java.lang.Object, java.lang.Object);
    void resize(int);
    void transfer(java.util.HashMap$Entry[]);
    public void putAll(java.util.Map);
    public java.lang.Object remove(java.lang.Object);
    final java.util.HashMap$Entry removeEntryForKey(java.lang.Object);
    final java.util.HashMap$Entry removeMapping(java.lang.Object);
    public void clear();
    public boolean containsValue(java.lang.Object);
    public java.lang.Object clone();
    void addEntry(int, java.lang.Object, java.lang.Object, int);
    void createEntry(int, java.lang.Object, java.lang.Object, int);
    java.util.Iterator newKeyIterator();
    java.util.Iterator newValueIterator();
    java.util.Iterator newEntryIterator();
    public java.util.Set keySet();
    public java.util.Collection values();
    public java.util.Set entrySet();
    int capacity();
    float loadFactor();
}
```

Constructors in HashMap:

(1) `HashMap h = new HashMap()`

Creates an empty HashMap object with default initial capacity is 16. Default fill ratio 0.75, like HashSet.

(2) `HashMap h=new HashMap(int initial capacity)`

(3) `HashMap h=new HashMap(int initial capacity, float fillratio).`

(4) `HashMap h=new HashMap(Map m);`

LinkedHashMap:

- This is similar to HashMap.

- The underlying data structure is HashTable+LinkedList.
- Insertion order is preserved.
- Introduced in java 1.2.

```

package collection;

import java.util.Iterator;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

public class CollectionDemo {
    public static void main(String[] args) {
        /*HashMap hm = new HashMap();
        hm.put(101,"ram");
        hm.put(102,"sam");

        System.out.println(hm);
        Set s = hm.keySet();
        System.out.println(s);
        Iterator i = s.iterator();
        while(i.hasNext()){
            System.out.println(i.next());
            hm.put(111,222);
        }*/
        ConcurrentHashMap hm = new ConcurrentHashMap();
        hm.put(101,"ram");
        hm.put(102,"sam");
        System.out.println(hm);
        Set s = hm.keySet();
        System.out.println(s);
        Iterator i = s.iterator();
        while(i.hasNext()){
            System.out.println(i.next());
            hm.put(111,222);
        }
        System.out.println(hm);
    }
}

```

Methods in LinkedHashMap:

```

$javap -java.util.LinkedHashMap
Compiled from "LinkedHashMap.java"
public class java.util.LinkedHashMap extends java.util.HashMap implements java.util.Map {
    public java.util.LinkedHashMap(int, float);
    public java.util.LinkedHashMap();
    public java.util.LinkedHashMap(Map);
    public java.util.LinkedHashMap(int, float, boolean);
    void init();
    void transfer(java.util.Map$Entry[]);
    public boolean containsValue(java.lang.Object);
    public java.lang.Object get(java.lang.Object);
    public void clear();
    java.util.Iterator newKeyIterator();
    java.util.Iterator newValueIterator();
    java.util.Iterator newEntryIterator();
    void addEntry(int, java.lang.Object, java.lang.Object, int);
    void createEntry(int, java.lang.Object, java.lang.Object, int);
    protected boolean removeEldestEntry(java.util.Map$Entry);
    static boolean access$000(java.util.LinkedHashMap);
    static java.util.LinkedHashMap$Entry access$100(java.util.LinkedHashMap);
}

```

Hashtable:

- Hashtable is similar to HashMap, which holds elements in the form of key-value pairs.
- It is a synchronized class.
- It is an thread safe.
- Performance is low, when compare to HashMap.
- Null value cannot be allowed in the place of key and value.
- It has introduced in java 1.0.
- The underlying data structure is Hashtable.
- Insertion order is not preserved and it is based on hashCode of an objects.
- Duplicate keys will not be allowed, but duplicate values can be allowed.
- Heterogeneous objects will be allowed for both key and values.

Constructors in Hashtable:

- (1) Hashtable ht=new Hashtable();

Initial capacity is 11 and default fillratio 0.75.

- (2) Hashtable ht=new Hashtable(int initialcapacity);
- (3) Hashtable ht=new Hashtable(int initialcapacity, float fillratio);
- (4) Hashtable ht=new Hashtable(map m);

properties	HashMap	Hashtable
version: synchronized: or Threadsafe	1.2 No	1.0 Yes
performance null keys null values default size loadfactor	high one multiples 16 0.75	low zero zero 11 0.75

IdentityHashMap:

In the case of duplication values in IdentityHashMap, the JVM will be uses the “==” operator for identifies the duplicates.

Where as in HashMap, the JVM will uses equals() method to identifies the duplicates.

WeakHashMap:

- It is exactly similar to HashMap.
- HashMap dominates GarbageCollector that is if any object associated with HashMap it is not eligible for Garbage Collector even though it doesn't have external references.
- But in the case of WeakHashMap an object is eligible for Garbage Collector.

SortedMap interface:

It is child interface of the Map.

If we want represents all the entries according to default sorting order of keys, then we should go for SortedMap.

Methods in SortedMap:

```
F:\>javap java.util.SortedMap
Compiled from "SortedMap.java"
public interface java.util.SortedMap extends java.util.Map{
    public abstract java.util.Comparator comparator();
    public abstract java.util.SortedMap subMap(java.lang.Object, java.lang.Object);
    public abstract java.util.SortedMap headMap(java.lang.Object);
    public abstract java.util.SortedMap tailMap(java.lang.Object);
    public abstract java.lang.Object firstKey();
    public abstract java.lang.Object lastKey();
    public abstract java.util.Set keySet();
    public abstract java.util.Collection values();
    public abstract java.util.Set entrySet();
}
```

Here Comparator comparator() method will return null if sorting order is not an default sorting order.

NavigableMap:

It the child interface of SortedMap.

It having some methods like below:

```
F:\>javap java.util.NavigableMap
Compiled from "NavigableMap.java"
public interface java.util.NavigableMap extends java.util.SortedMap{
    public abstract java.util.Map$Entry lowerEntry(java.lang.Object);
    public abstract java.lang.Object lowerKey(java.lang.Object);
    public abstract java.util.Map$Entry floorEntry(java.lang.Object);
    public abstract java.lang.Object floorKey(java.lang.Object);
    public abstract java.util.Map$Entry ceilingEntry(java.lang.Object);
    public abstract java.lang.Object ceilingKey(java.lang.Object);
    public abstract java.util.Map$Entry higherEntry(java.lang.Object);
    public abstract java.lang.Object higherKey(java.lang.Object);
    public abstract java.util.Map$Entry firstEntry();
    public abstract java.util.Map$Entry lastEntry();
    public abstract java.util.Map$Entry pollFirstEntry();
    public abstract java.util.Map$Entry pollLastEntry();
    public abstract java.util.NavigableMap descendingMap();
    public abstract java.util.NavigableSet navigableKeySet();
    public abstract java.util.NavigableSet descendingKeySet();
    public abstract java.util.NavigableMap subMap(java.lang.Object, boolean, java.lang.Object, boolean);
    public abstract java.util.NavigableMap headMap(java.lang.Object, boolean);
    public abstract java.util.NavigableMap tailMap(java.lang.Object, boolean);
    public abstract java.util.SortedMap subMap(java.lang.Object, java.lang.Object);
}
    public abstract java.util.SortedMap headMap(java.lang.Object);
    public abstract java.util.SortedMap tailMap(java.lang.Object);
```

TreeMap:

- TreeMap if follows the one datastructure is called as RED-BLACK tree.
- Insertion order is not preserved but all the objects can be arranged according some sorting order of keys. Hence sorting order is preserved.

- Duplicate values cannot allow for keys, but values can be duplicate.
- If we use default sorting order, then the keys must be homogeneous and class must be implement Comparable interface, otherwise we will get a ClassCastException.
- If we use customized sorting order, then the keys need not be homogenous (heterogeneous) and class must be implements Comparator interface.

Null acceptance:

- For the empty TreeMap as the first element with null key is allowed, but after inserting null entry, we should not try to enter another null entries, if done we will get NullPointerException.
- For Non-empty TreeMap, we should not try enter any null entries, if done, we will get NullPointerException.
- There are no restrictions for values about null acceptance.
- If we not provide any sorting order then by default sorting order will be used.

Constructors in TreeMap:

- (1) `TreeMap t=new TreeMap();`
- (2) `TreeMap t=new TreeMap(Comparator c);`
- (3) `TreeMap t=new Treemap(Map m);`
- (4) `TreeMap t=new TreeMap(SortedMap s);`

Properties:

It is the child class of Hashtable class.

It can be used for representing key-values pairs.

Both key and values should be String objects.

Properties p =new Properties();

Methods in Properties class:

- (1) `String getProperty(String name):`

Returns the value associated with specified property.

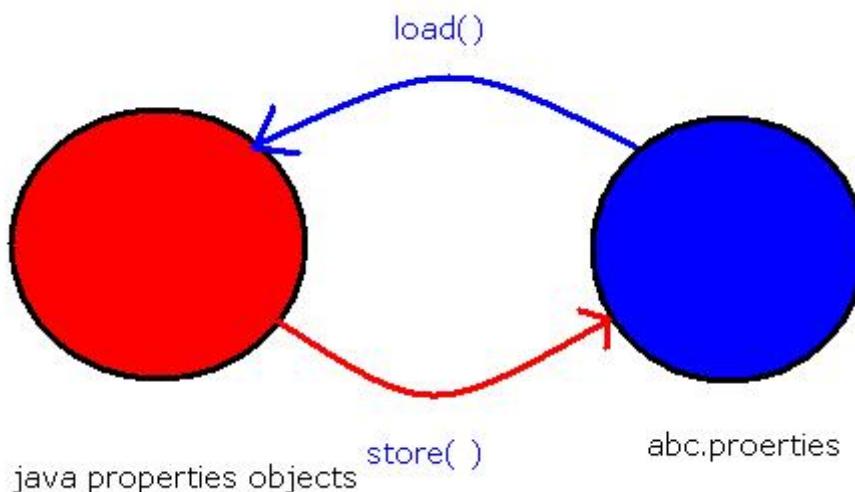
If the specified property is not available, then it returns null values.

- (2) String setProperty(String name, String value);
- (3) Enumeration getPropertyNames();
- (4) Void load(InputStream is)

To load the properties from properties file to java properties object.

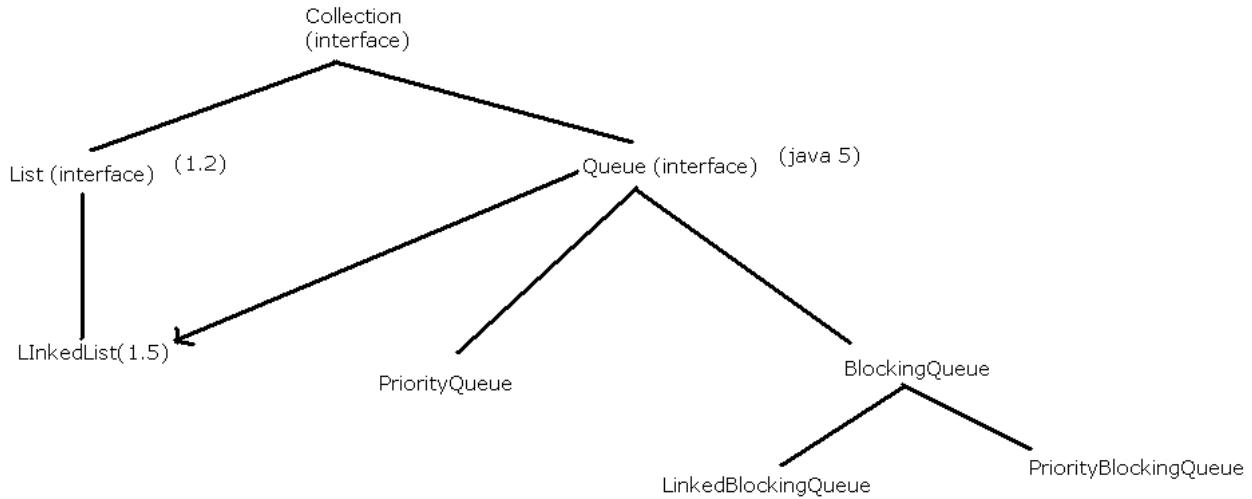
- (5) Void store(OutputStream os, String comment);

To store the properties from java properties object to properties file.



Queue Interface:

- If we want to represent a group of individual object prior to processing, then we should go for Queue interface.
- Queue in general follows FIFO order, but we can implement our own order by using PriorityQueues.
- LinkedList class is re-engineered in java5 version to implement Queue interface.
- LinkedList based implementation of queue always follows FIFO.



Methods in Queue:

```

javap java.util.Queue
Compiled from "Queue.java"
public interface java.util.Queue extends java.util.Collection<
    public abstract boolean add(java.lang.Object);
    public abstract boolean offer(java.lang.Object);
    public abstract java.lang.Object remove();
    public abstract java.lang.Object poll();
    public abstract java.lang.Object element();
    public abstract java.lang.Object peek();
>

```

PriorityQueue:

- If we want to store the objects prior to processing according to some priority, then we should go for PriorityQueue.
- Insertion order is not preserved. But all the elements are arranged according to some priority. It may be default natural storing order or customized sorting order specified by Comparator object.
- If your are depend upon the natural sorting, then elements should be homogeneous and comparable otherwise we will get ClassCastException.
- If we are depended upon the our own sorting order then elements need be homogeneous and comparable.
- Duplication objects are not allowed.
- Null insertion is not possible even the first element itself.

Constructors in PriorityQueue:

- (1) PriorityQueue pq=new PriorityQueue();

Creates a default Priority Queue, with the default size is 11 and all objects are sorted according some sorting order.

- (2) `PriorityQueue pq=new PriorityQueue(int initialcapacity , comparatory c);`
- (3) `PriorityQueue pq=new PriorityQueue(SortingSet s);`
- (4) `PriorityQueue pq=new PriorityQueue(Collection c);`

Collections:

Collections is a utility class, provides several utility methods for the collection implemented classes.

(A) Sorting a List:

“Collections” class having a method to sorting a list i.e.,

`Public static void sort (List L);`

This method will sort the element of List by using some default sorting order.

If we use default sorting order, the elements must be homogeneous and comparable, otherwise we will get ClassCastException.

The List should not contain null values; otherwise we will get a NullPointerException.

`Public void static sort (List l, comparator c)`

This method will be useful for sorting the elements of List with some customized sorting order; here the elements need not homogeneous and comparable.

(2) Searching the List:

Public static int binarySearch(List l, Object key);

This method will be useful for searching the List elements with default searching order.

It is internally follows binarysearch algorithm.

Before using this method, the List elements should be sorted, otherwise we may get unreliable values.

The successful searching, this method will returns index value.

Otherwise it will show insertion place/index.

Insertion place/index is the place where we should enter some value to proper sorted.

Public static int binarySearch (List I, Object key, Comparator c);

This method will be useful for in the case of List is sorted by using Comparator.

We should pass the same Comparator object when we going to search the element from List.

Reversing the List elements:

Collection class contains one method for reversing the elements of List.

Public static void reverse (List I);

Arrays class:

Java.util package contain one utility class called Arrays.

This will provide some utility methods.

Sorting the elements of Array:

Following methods are helpful in sorting the Array elements.

Public static void sort(primitives[] p);

This method will sort the primitive elements of an array,

It will follow the default sorting order.

Public static void sort(Object[] o);

This method will sort the object type of elements of an Array..

It will maintain the default sorting order.

Public static void sort(Object[] o, comparator c);

To Sort the object Array according to customized sorting order.

Note:

The object Arrays will sort either customized sorting order or default sorting order, whereas primitive Arrays will sort only by natural/default sorting order only and not by customized sorting order.

Searching an Array:

- (a) ***Public static int binarySearch(primitive[] p, primitive key);***
- (b) ***Public static int binarySearch(Object[] o, Object key);***
- (c) ***Public static int binarySearch(Object[] o, Object key, Comparator c);***

Converting of Arrays as a List:

- Arrays class contains one method to convert the Array in to List, i.e..
- Public static List asList(Objective[] o);
- This method won't create a List object, the Array object will be given in the form of List object.
- By using Array object, if we done any modification automatically it will reflect to List and same time if we done modification on List reference automatically it will reflect on Array object.
- By using List reference if we are performing any operation which varies the size we will get runtime error saying:
UnsupportedOperationException.

Random Class:

```
public class CollectionDemo {  
    public static void main(String[] args){  
        PriorityQueue pq = new PriorityQueue();  
        Random r = new Random();  
        int count = 0;
```

```

while(true){

    int i = r.nextInt(1000);

    if(200<i && i<300){

        System.out.println(i);

        count++;

        pq.offer(i);

    }

    if(count==10){

        break;

    }

    //System.out.println(" " +i);

}

}

}

```

Data and SimpleDateFormat:

```

import java.text.SimpleDateFormat;
import java.util.Date;
public class Demo{
    public static void main(String[] args){
        Date d = new Date();
        System.out.println(d);
        //SimpleDateFormat sdf = new SimpleDateFormat();
        //SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        //SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yy");
        //SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yy");

        //SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yy");
        //SimpleDateFormat sdf = new SimpleDateFormat("ddd-MMM-yy");
        SimpleDateFormat sdf = new
            SimpleDateFormat("dd-MMM-yy hh:mm:ss");
        String s = sdf.format(d);
        System.out.println(s);
    }
}

```

```
}
```

Generic Programs:

Generics:

If we go for arrays, by default arrays are providing type safety, that it allows only homogeneous data.

```
int a[] = new int[5];  
  
a[0]=10;  
a[1]=(byte)20;  
a[2]=(short)30;  
a[3]='a';  
a[4]=12.09;//ce
```

When ever we reading the data from array no need to do type casting.

But if we go for old collection, there is no type safety that means it can allows all the type of data.

Whenever we reading the data from collection object we need to do downcasting, if we are not doing downcasting properly we need to face one runtime exception that is `java.lang.ClassCastException`

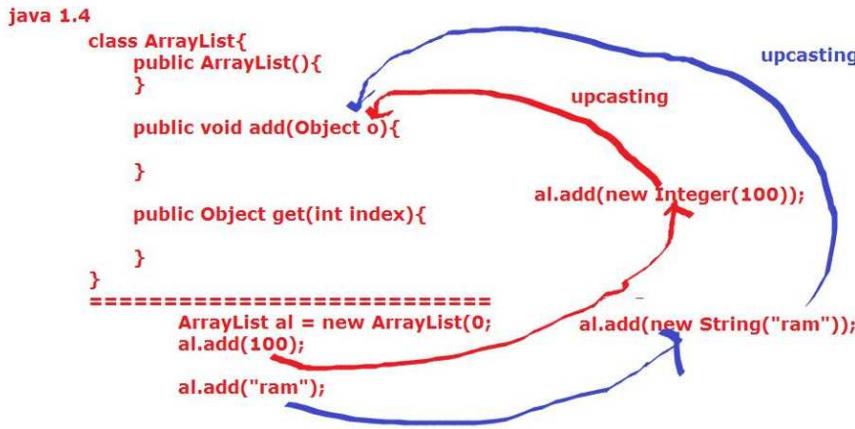
But if we go for old collection, there is no type safety that means it can allows all the type of data.

Whenever we reading the data from collection object we need to do downcasting, if we are not doing downcasting properly we need to face one runtime exception that is `java.lang.ClassCastException`

```
ArrayList al = new ArrayList();  
al.add(10);  
al.add(20);  
al.add("ram");  
al.add(false);  
System.out.println(al);  
for(Object o1: al){  
    Integer i = (Integer)o1;  
    System.out.println(i);  
}  
}  
o/p: 10 20  
java.lang.ClassCastException: java.lang.String can not convert into  
java.lang.Integer
```

To avoiding old Collection class drawbacks we should go for Generic

Generic is combination of collection class functionalities + Type Safety



--> By default arrays provides typesafety.
 --> So no need to do downcasting while reading the data
 from array.

 --> CollectionsFramework does not provides typesafety, so
 we need to do downcasting.

In the meanwhile of downcasting we are going to be face `java.lang.ClassCastException`.

To avoids above drawbacks we should go for Generic.

```

Generic
CollectionsFramework Functionaliteis
+
Type Safety

--> It was introduced in the java 1.5 version

```

```

package generic;

class Test<T>{

    T i;

    Test(T i){

        this.i = i;
    }

    T getValue(){

        return i;
    }
}

```

```
public class GenericDemo {  
    public static void main(String[] args) {  
        Test<Integer> t = new Test<Integer>(10);  
        System.out.println(t.getValue());  
        Test<String> t1 = new Test<String>("Nikhil");  
        System.out.println(t1.getValue());  
    }  
}
```

m1(ArrayList<? extends X>){}
if X is class--> ? will be replaced with x data or its sub
class data
if X is interface--> ? will be replaced with x data or its
implementation class data

m1(ArrayList<? super X>){}
if X is class--> ? will be replaced with x data or its super
class data
if X is interface--> ? will be replaced with x
implementation class super class

m1(ArrayList<?>){}
? will be replaced with all type of data

m1(ArrayList<Object>){}
? will be replaced with only Object reference type of data

```
package generic;  
  
class Test<T extends Number>{  
    T i;  
    Test(T i){  
        this.i = i;  
    }  
    T getValue(){  
        return i;  
    }
```

```
}

public class GenericDemo {

    public static void main(String[] args) {

        Test<Integer> t = new Test<Integer>(10);

        System.out.println(t.getValue());

        /*Test<String> t1 = new Test<String>("Nikhil");

        System.out.println(t1.getValue());*/

        Test<Float> t1 = new Test<Float>(20.0f);

        System.out.println(t1.getValue());

    }

}

=====

package generic;

class Test<T extends Runnable>{

    T i;

    Test(T i){

        this.i = i;

    }

    T getValue(){

        return i;

    }

    void execute(){

        Thread t = new Thread(i);

        t.start();

    }

}
```

```
    }

}

class MyThread implements Runnable{

    public void run(){

        System.out.println("run method");

    }

}

public class GenericDemo {

    public static void main(String[] args) {

        Test<Thread > t = new Test<Thread>(new Thread());

        System.out.println(t.getValue());

        MyThread mt = new MyThread();

        Test<MyThread> t1 = new Test<MyThread>(mt);

        System.out.println(t1.getValue());

        t1.execute();

    }

}

=====

package generic;

import java.util.ArrayList;

class A{

    int a =111;

}

class B extends A{
```

```
int b = 222;  
}  
  
public class GenericDemo {  
  
    static void m1(ArrayList<? extends Number> al){  
  
        System.out.println("m1 method");  
  
        System.out.println(al);  
    }  
  
    public static void main(String[] args) {  
  
        ArrayList<Integer> al = new ArrayList<Integer>();  
  
        al.add(100);  
  
        al.add(200);  
  
        //al.add("ram");  
  
        m1(al);  
  
        ArrayList<Double> al2 = new ArrayList<Double>();  
  
        al2.add(100.00d);  
  
        al2.add(200.00d);  
  
        m1(al2);  
  
        ArrayList<String> al1 = new ArrayList<String>();  
  
        al1.add("ram");  
  
        al1.add("kiran");  
  
        //al1.add(false);  
  
        //m1(al1);  
    }  
}
```

```
=====
package generic;

import java.util.ArrayList;

class A{

    int a;

    A(int a){

        this.a = a;

    }

    A(){}

    public String toString(){

        return a+" ";

    }

}

class B extends A{

    String b;

    B(int a , String b){

        super();

        this.a = a;

        this.b = b;

    }

    public String toString(){

        return a+"...."+b;

    }

}
```

```
public class GenericDemo {  
    static void m1(ArrayList<?> al){  
        System.out.println("m1 method");  
        System.out.println(al);  
    }  
    public static void main(String[] args) {  
        ArrayList<Integer> al1 = new ArrayList<Integer>();  
        al1.add(10);  
        al1.add(20);  
        al1.add(30);  
        m1(al1);  
        ArrayList<String> al2 = new ArrayList<String>();  
        al2.add("ram");  
        al2.add("suji");  
        m1(al2);  
    }  
}
```

enum:

enum is used grouping the named constants.

With the help of enum we can develop user define datatypes.

It has been introduced in java 1.5 version.

Ex:

```
enum Color{  
    BLUE, GREEN, YELLOW;  
}
```

enum is always followed by enum_name.

We can write empty enums also.

Ex:

```
enum Color{  
}
```

Naming conventions of enum same as class.

Ex:

```
enum Month{  
    jan,feb,march;  
}
```

We can write the enum constants either in small letters or in capital letters or in combinations.

Constants may ended with the semicolon(;). It not a mandatory.

Ex:

```
enum Tifin{  
    Idly,Dosa  
}
```

If we want write any other content otherthan constants the constants must be ended with semicolon.

Ex:

```
enum Tifin{  
    Idly,Dosa; //here ';' is mandatory.  
    void m1(){  
    }  
}
```

We cannot write abstract methods in constants.

```
enum Tifin{  
    Idly;  
    //abstract void m1();  
}
```

If we want to write any other content without constants in enum, before the content we should be place semicolon(;) .

Ex:

```
enum Tifin{  
    ; //mandatory  
    void m1(){  
    }  
}
```

Within the enum we can write main method, we can compile and execute the programs.

Without class definition we can write the java code we can compile and execute. For this purpose we have two ways.

1. with enum

2. with interface

1. with enum (from java 1.5 onwards)

```
enum Color{  
    ;  
    public static void main(String args[]){  
        System.out.println("we can");  
    }  
}
```

2. with interface (from java 1.8)

```
interface I{  
    public static void main(String args[]){  
        System.out.println("we can");  
    }  
}
```

```
    }  
}  
}
```

We can write the enums outside of the class and inside of the class.

If we are writing enum outside of the class, the below modifiers are allowed.

Ex: public, package-private (default), strictfp

If we are writing enum inside of the class, the below modifiers are allowed.

Ex: public, package-private (default), strictfp, private, protected, static, but we can not write final.

In above two conditions we can not write abstract keyword.

Every user define enum is by default subclass of java.lang.Enum.

We cannot write extends keyword after enum name, but we can write implements keyword.

Ex:

```
enum Color extends Object{//wrong syntax}
```

```
}
```

Ex:

```
enum Color implements Runnable{
```

```
;
```

```
    public void run(){
```

```
}
```

```
}
```

From java 1.5 onwards we can use enum constants value in the switch case also.

"case" variables names must be equal or less than enum constants, and spelling must be equal (including lower and upper case).

We can not create object for enum.

But we can create reference for enum.

Ex:

```
enum Color{  
    ;  
    public static void main(String[] ram){  
        //Color c = new Color; //wrong syntax  
        Color c;  
    }  
}
```

With the help of enum reference we can hold constant values.

If we want read values from enum, we have one method like values().

If we want know the positions of constants, then we have one method like ordinal().

Ex: class A{

```
enum Color{  
    Blue,Yellow,Red;  
}  
public static void main(String... ram){  
    Color c[] = Color.values();  
    for(Color c1 : c){
```

```
S.o.p(c1+"..."+c1.ordinal());  
    }  
}  
}  
  
package enums;  
  
enum Color{  
    BLUE, GREEN, YELLOW, red  
}  
  
enum Flower{  
    ;  
    void m3(){  
    }  
}  
  
/*enum A extends java.lang.Object{  
} */  
  
enum B implements java.lang.Runnable{  
    ;  
    public void run(){  
    }  
}  
  
public class EnumDemo {  
    enum Month{  
        Jan, Feb;  
        void m1(){
```

```
}

//abstract void m2();

}

public static void main(String[] args) {

    //Color c = new Color;

    Color c[] = Color.values();

    for(Color c1: c){

        System.out.println(c1+"..."+c1.ordinal());

    }

}

}
```

enum constants by default public static final.

enum with switch case:

```
enum Color{

    BLUE, GREEN, YELLOW, RED;

}

public class EnumDemo {

    public static void main(String[] args) {

        Color c = Color.YELLOW;

        switch(c){

            case BLUE: System.out.println("THIS IS BLUE COLOR");

            break;

        }

    }

}
```

```

        case GREEN: System.out.println("THIS IS GREEN COLOR");
                    break;

        case YELLOW: System.out.println("THIS IS YELLOW COLOR");
                    break;

/*case RED: System.out.println("THIS IS RED COLOR");
                    break;*/

/*case PINK: System.out.println("THIS IS PINK COLOR");
                    break;*/

        default : System.out.println("NO COLOR MATCHED");

    }}}
```

/*switch statement case values must same as enum constants. We cannot use other than enum constant values, but we can decrease the enum constants.*/

```

import java.util.Scanner;
```

```

public class EnumDemo {
    enum Month{
        jan(31),feb,march(31);
        int noofdays;
        Month(){
            Scanner scan = new Scanner(System.in);
            System.out.println("enter days");
            int x = scan.nextInt();
            //noofdays=28;
            noofdays = x;
        }
        Month(int days){
            noofdays = days;
        }
        int getDays(){
            return noofdays;
        }
    }
    public static void main(String[] r){
        Month[] m = Month.values();
        for(Month m1: m){
            System.out.println(m1+"..."+m1.getDays());
        }
    }
}
```

```
    }  
}
```

Socket Programming:

Network: Interconnection between any two systems with in local or remote is called network.

Types of network:

- 1) LAN
- 2) MAN
- 3) WAN

LAN: Stands for Local Area Network.

The systems which are connected with in the organization level or building level is called LAN.

MAN: Stands for Metropolitan Area Network.

The systems which are connected with in the city level are called MAN.

WAN: Stands for Wide Area Network.

The systems which are connected throughout the world level are called WAN.

Request: The data (input) which is need to processing is called request.

Response: The data (output) which has processed is called response.

Client Program: The program which is able to read the input values and send to another program is called client program.

Client Machine: The machine which has client program is called Client Machine.

Server program: The program which is able to send output values to client program is called server program.

Server Machine: The machine which has server program is called Server machine.

IP Address: Every system is identified in the network with one unique identification number that identification number is called IP Address. No two systems have same IP Address.

IP Address starts with 0.0.0.0 to 255.255.255.255

Hostname: The alternative of IP Address is called Hostname.

We can identify the system either through IP Address or hostname.

Port: Every services running on unique number, that number is called port. No two services have same port number, if have those two services not running at a time.

Socket: It is a listener, which is used to send and read the request and response.

Client side listener is called Socket.

Server side listener is called ServerSocket

Protocol: It is a set of instructions. It is used to transform the data from client machine to server machine.

Types of protocols:

TCP/IP: Transfer Control Protocol/Internet Protocol

UDP: User Datagram protocol.

Under the TCP we have different sub protocols. They are

- 1) http
- 2) https
- 3) smtp
- 4) ftp

URL: Unified Resource Locator. It is used to identify the resource location. It provides absolute path.

Absolute path means combination of

Protocol name + hostname/IP address + port number + resource + query string

URI: Unified Resource Identifier. It is directly represent the resource and query string.

It provides relative path.

```
Socket s = new Socket("localhost",7777);
DataOutputStream dos = new DataOutputStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
BufferedReader br1 = new BufferedReader(new InputStreamReader(s.getInputStream()));
String str,str1;
while(!(str=br.readLine()).equals("exit")){
    dos.writeBytes(str+"\n");str1=br1.readLine();System.out.println(str1);
}
=====
ServerSocket ss = new ServerSocket(7777);Socket s=ss.accept();System.out.println("connection
established");
PrintStream ps = new PrintStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
BufferedReader br1 = new BufferedReader(new InputStreamReader(s.getInputStream()));
String str,str1;
while(true){
    while((str=br1.readLine())!=null){
        System.out.println(str);str1=br.readLine();
        ps.println(str1);
    }
}
```

ClientDemo.java:

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
public class ClientDemo {
    public static void main(String[] args) throws UnknownHostException,
IOException {
        Socket s = new Socket("localhost",7777);
        DataOutputStream dos= new
DataOutputStream(s.getOutputStream());
```

```
    BufferedReader br = new BufferedReader(new  
InputStreamReader (System.in));  
  
    BufferedReader br1 = new BufferedReader(new  
InputStreamReader (s.getInputStream()));  
  
    String str,str1;  
  
    while(!(str=br.readLine()).equals("exit")){  
  
        dos.writeBytes(str+"\n");  
  
        str1=br1.readLine();  
  
        System.out.println(str1);  
  
    }  
  
}
```

ServerDemo.java

```
import java.io.BufferedReader;  
  
import java.io.IOException;  
  
import java.io.InputStreamReader;  
  
import java.io.PrintStream;  
  
import java.net.ServerSocket;  
  
import java.net.Socket;  
  
public class ServerDemo {  
  
    public static void main(String[] args) throws IOException {  
  
        ServerSocket ss = new ServerSocket(7777);  
  
        Socket s=ss.accept();  
  
        System.out.println("connection established");  
  
        PrintStream ps = new PrintStream(s.getOutputStream());
```

```
BufferedReader br = new BufferedReader(new InputStreamReader
(System.in));

    BufferedReader br1 = new BufferedReader(new
InputStreamReader (s.getInputStream()));

    String str,str1;

    while(true){

        while((str=br1.readLine())!=null){

            System.out.println(str);

            str1=br.readLine();

            ps.println(str1);

        }

    }}}
```

Reflection API:

If we want know source code information from ".class" file(byte code) then we can go for Reflection API.

With reflection API we can read information from runtime loaded class(.class).

Reflection API provides mirror information about variables, methods, constructors, annotations, exception class information.

With the help of reflection API we can access both private and public data.

This Reflection API is coming under
java.lang.reflect package.

Important classes under java.lang.reflect packages are

- 1)Filed
- 2)Constructor

3)Method

4)Modifer

Filed: It is used to store the variable information. Variables may public or private.

Method: It is used to store the methods information (public or private)

Constructor: It is used to store the information about constructors.

Modifier: It is used to store information about access modifier and modifier.

This class provides the modifier information in the form of int.

If want convert from int to string then Modifier class itself provides some predefine method (`toString()`).

ABSTRACT	1024
FINAL	16
INTERFACE	512
NATIVE	256
PRIVATE	2
PROTECTED	4
PUBLIC	1
STATIC	8
STRICT	2048
SYNCHRONIZED	32
TRANSIENT	128
VOLATILE	64

java.lang.Class: This class having capability to hold bytecode information.

Important method under `java.lang.Class`:

1)`getDeclaredFields()`

- 2) getFields()
- 3) getDeclaredMethods()
- 4) getMethods()
- 5) getDeclaredConstructors()
- 6) getConstructors()

forName():

It is an one static factory method, is used to loads the byte code of any class.

forName() always needs bytecode not source code.

Syntax : Class cls = Class.forName("class_name");

This method has been throwing one compile time exception i.e

Java.lang.ClassNotFoundException.

forName() is throwing exception like ClassNotFoundException if we are not passing exsisting ".class" file name(classname).

newInstance():

This is one instance factory methods, which is used to create object with the help of class reference variable.

Object obj = cls.newInstance();

newInstance() has been throwing two compile time exceptions.

- 1) IllegalAccessException
- 2) InstantiationException

Jvm will throwing Instantiation Exception, if given ".class " file(class) doesnot contains zero argument constructor.

Private variables by defaultly having one background method like setAccessible(false), so this method doesnot provide permission to access the data out side of the class.

If we want to access private data out side of the class we need to convert from setAccessible(false) to setAccessible(true), otherwise we will IllegalAccessException

```
import java.io.FileNotFoundException;
import java.io.IOException;
public class A {
    public int b=2000;
    A(){    //getDeclatedConstructor or Constructor
        System.out.println("A class construc");
    }
    A(int x){
        System.out.println("A class paramconstruc");
    }
    private static int c =3000;
    int m1()throws IOException,FileNotFoundException{
        return 10;
    }
    public void m2(int x){
        System.out.println("hi");
    }
}

import java.lang.reflect.Constructor;
```

```
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
public class B extends A{
    public static void main(String[] args)
        throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        java.lang.Class cls = java.lang.Class.forName("A");
        java.lang.Object obj = cls.newInstance();
        Field f[] = cls.getDeclaredFields();
        for(Field f1:f){
            String modi = Modifier.toString(f1.getModifiers());
            if(modi.contains("private")){
                System.out.println(f1);
                f1.setAccessible(true);
                System.out.println(f1.get(obj));
                Object type = f1.getType();
                System.out.println(type);
            }
            System.out.println(f1);
        }
        System.out.println("*****");
        Constructor[] c = cls.getDeclaredConstructors();
        for(Constructor c1:c){
```

```
        System.out.println(c1);

    }

    System.out.println("*****");
    Method[] m = cls.getDeclaredMethods();
    for(Method a:m){

        System.out.println(a);
        System.out.println("*****");
        Class[] e = a.getExceptionTypes();
        for(Class e1:e){

            System.out.println(e1);
        }
    }

}
```