CORE JAVA

Programming language

A language which is used to communicate between user and computer is called programming language.

Programming language acts like a mediator or interface betwen user and computer.

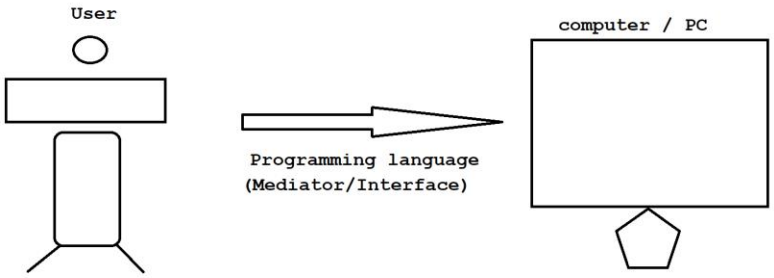


Diagram: introduction1.1

Java

Object oriented programming language.
Platform independent programming language.
Case sensitive programming language
Strongly typed checking language.
High level programming language.
Open source programming language

1995 --> James Gosling --> Sun Micro system --> Oracle Corporation Java software --> JDK software

C language

Procedure oriented programming language.
Platform dependent programming language.
Case sensitive programming language
Lossely typed checking language.
Middle level language (LOW + HIGH)

Interview Questions

Q) What is Java?

Java is a object oriented, platform independent, case sensitive, strongly typed checking, high level, open source programming language developed by James Gosling in the year of 1995.

Programming language

A language which is used to communicate between user and computer is called programming language.

Programming language acts like a mediator or interface between user and computer.

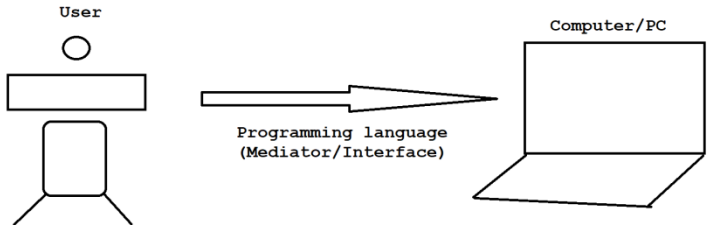


Diagram: introduction2.1

Programming language is divided into two types.

1. Low Level Language
2. High Level Language

1. Low Level Language

A language which is understand by a computer easily is called low level language.
A language which is computer dependent is called low level language.

ex: Machine language
Assembly language

Machine language

It is a fundamental language of a computer which is combination of
0's and 1's.
It is also known as binary language.
A computer understands many languages but to understand machine language computer does not required any translator.

Advantages:

> A program writtens in machine language consumes less memory.
> It does not required any translator.
> It is more efficient when compare to other languages.

Disadvantages:

> It is a burden on a programmer to remember dozen's of binary code.
> If anywhere error raised in our program then locating and handling that becomes difficult.
> Modifications we can't do easily.

Assembly language

The second generation language came into an existence is called assembly language.
Assembly language is a replacement of symbols and letters for mathematical programming code .i.e opcode values.
It is also known as symbolic language.
Assembly language can't understand by a computer directly.We required translator.
We have three translators.

1)Assembler
2)Compiler
3)Interpreter

1) Assembler

It is one of the translator which is used to convert assemblic code to machine code.

Merits:

> If anywhere error raised in our program then locating and handling that error becomes easy.

> Modifications can be done easily.

Demerits:

> It is a mind trick to remember all symbolic code.

> It requires translator.

> It is less efficient when compare to machine language.

Q)What is Debugging?

Bug is also known as Error.

The process of eliminating the bugs from the application is called debugging.

2. High Level Language

A language which is understand by a user easily is called high level language.

A language which is user dependent is called high level language.

ex: C++, C#, Java, .Net, Python and etc.

High level language can't understand by a computer easily.We required translator.

compiler It will compile and execute our program at a time.

Interpreter It will execute our program line by line procedure.

Advantages:

> It is easy to learn and easy to use because it is similar to english language.

> Debugging can be done easily.

> Modifications can be done easily.

Disadvantages:

> A program writtens in high level language comes huge amount of memory.

> It requires translator.

> It is not efficient when compare to low level language.

Escape Characters or Escape Sequences

Escape characters are used to design our output in neat and clean manner.

All escape characters starts with back slash(\) followed by a single characater.

ex: \n

Mostly escape characters are placed inside output statement in java.

ex: System.out.println("\n");

We have following escape characters or sequences in java.

1)\n (new line)

2)\t (horizontal tab)

3)\b (back space)

4)\r (carriage return)

5)\f (form feeding)

6)\\ (back slash)

7)\" (double quote)
8)\' (single quote) and etc.

1)\n (new line)

```
class Dhanush
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\nTALENT");
	}
}
```

o/p:

```
IHUB
TALENT
```

2)\t (horizontal tab)

```
class Chiranjeevi
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\tTALENT");
	}
}
```

o/p:

```
IHUB  TALENT
```

3)\b (back space)

```
class Divya
{
	public static void main(String[] args)
	{
		System.out.println("IHUBTAL\bENT");
	}
}
```

o/p:

```
IHUBTAENT
```

ex:

```
class Kalyani
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\b\b\bTALENT");
	}
}
```

o/p: ITALENT

4)\r (carriage return)

```
class Gopala
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\rTALENT");
	}
}
```

o/p:

```
	TALENT
```

ex:

```
class Karthik
{
	public static void main(String[] args)
	{
		System.out.println("TALENT\rIHUB");
	}
}
```

o/p:

```
	IHUBNT
```

6)\\ (back slash)

```
class Saritha
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\\TALENT");
	}
}
```

o/p:

```
	IHUB\TALENT
```

7)\" (double quote)

```
class Sufiyan
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\"TALENT");
	}
}
```

o/p:

```
	IHUB"TALENT
```

8)\' (single quote)

```
class Vyshnavi
{
	public static void main(String[] args)
	{
		System.out.println("IHUB'TALENT");
		System.out.println("IHUB\'TALENT");
	}
}
```

o/p: IHUB'TALENT
IHUB'TALENT

Interview question

Q)Write a c program to print %d ?

```
void main()
{
	clrscr();

	printf("%d");   // 0

	getch();
}
```

ex:

```
void main()
{
	clrscr();

	printf("%%d");		// %d

	getch();
}
```

Q)What is the output of below program?

```
class Demo
{
	public static void main(String[] args)
	{
		System.out.print("\nek");
		System.out.print("\boi");
		System.out.print("\rha");
	}
}
```

o/p:

hai

SDLC

SDLC stands for Software Development Life Cycle.
It is a process adopt by IT industry to develop accurate and quality of softwares.
There are six phases are there in SDLC.

1)Feasibility study
2)Analysis
3)Designing
4)Coding
5)Testing
6)Delivery and Maintainence

1)Feasibility study

Feasibility study completed depends upon TELOS formulea.

ex: T - Technical Feasibility
E - Ecnomical Feasibility
L - Legal Feasibility
O - Operational Feasibility
S - Scheduled Feasibility

All the above information they will keep in a document called BDD.
BDD stands for Business Designed Document.

2)Analysis

In this phase , system analyst or product owner will involved.
They will seperate system requirements and software requirements.
They will keep this information in a document called SRS.
SRS stands Software Requirement Specification.

3)Designing

Designing can be performed in two ways.
i)High level designing

Manager is responsible to perform high level designing.
In high level designing ,we will design main modules.

ii)Low level designing

Team Lead/Project Lead is resposible to perform low level designing.
In low level designing , we will design sub-module/child modules.
All this above information we will keep in a document called PDD/TDD.

Here PDD stands for Project Design Document.
Here TDD stands for Technical Design Document.

4)Coding

In coding phase, developers will involved.
Developers are responsible to generate the build(source code).
Once our build is ready then developer even responsible to perform white box testing.

5)Testing Phase
In this phase, testing team or QA team will involved.
The will test the code by using one software component called STLC.
STLC stands for Software Testing Life Cycle.
Testing Team or QA team is responsible to perform black box testing.

6)Delivery & maintainence phase
Before delivery we will perform UAT testing.
UAT stands for User Acceptance Testing.
UAT testing is classified into two types.

i)Alpha testing
ii)Beta testing

Project
Project is a collection of modules.
ex:

customer module
login module
registration module
payment module
report generation module
and etc.

Every project contains two domains.

1)Technical Domain

Using which technology we developed our project.
ex: Java

2)Functional Domain

It describes state of a project.
ex: Healthcare domain
Insaurance domain
Banking domain
ERP domain
and etc.

Modules In Java
There are three modules in java.

JAVA

JSE/J2SE	JEE/J2EE	JME/J2ME
(Java Standard Edition)	(Java Enterprise Edition)	(Java Micro Edition)
> Standalone App	> Distributed App	> Mobile App
> Desktop App	> Enterprise App	
> Two-tier App	> ERP App	
	> N-Tier App	

> Standalone App

A normal java program which contains main method is called standalone application.

ex:

```
class  Test
{
        public static void main(String[] args)
        {
                -
                -
        }
}
```

> Desktop App

It is a software application which is used to perform perticular task.

It is specially designed for laptops and PC's.

ex: Control panel, Recycle Bin

> Two-tier App

Having more then one tier is called two-tier application.

Diagram: java1.1

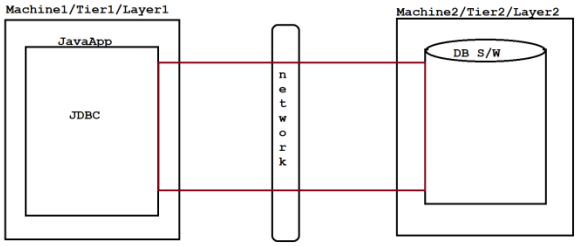
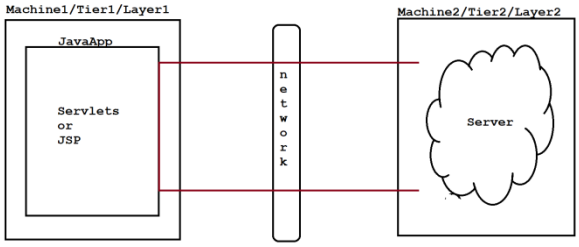


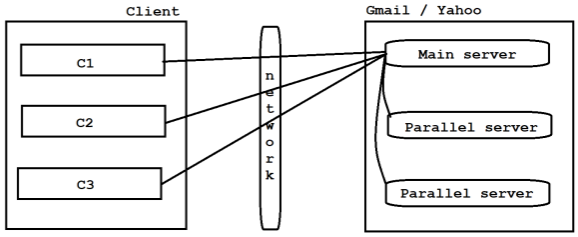
Diagram: java1.2



> Distributed App

In client-server architecture, if multiple clients are giving the request to the main server then main server will forward the request to it's parallel servers to reduce the burden of main server such type of application is called distributed application.

Diagram: java1.3



> Enterprise App

An application which deals with large business complex logic with the help of middleware services is called enterprises application.

Here midleware services means authentication, autherization, malware production, firewall, security and etc.

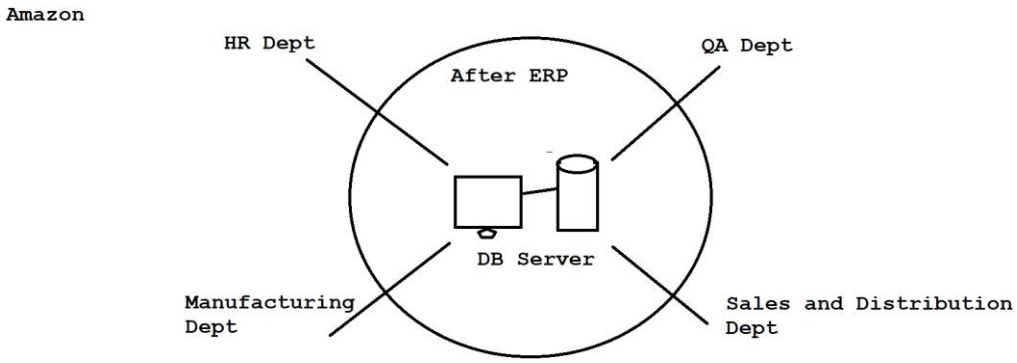
ex: facebook
Online shopping websites

> ERP App

ERP stands for Enterprise Resource Planning.

ERP application is used to maintain the data in a enterprise.

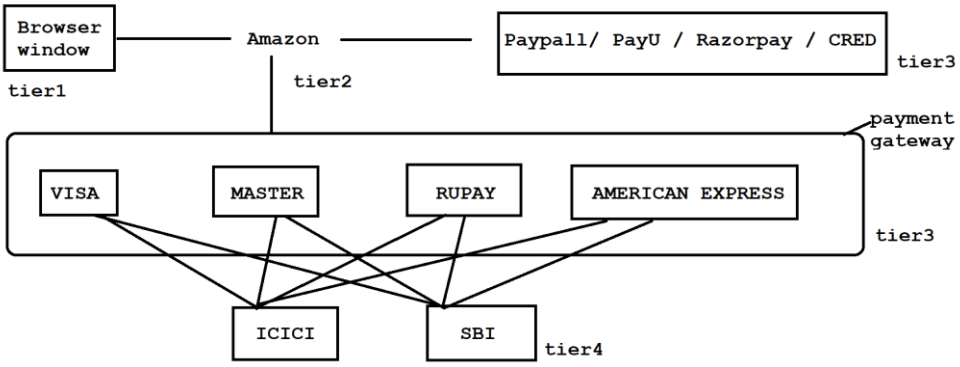
Diagram: java1.4



> N-Tier App

Having more then two tiers is called N-tier application.

Diagram: java1.5



> Mobile App

It is a software application or a program which is specially designed for wireless network devices like phone, tab, cell, celluar and etc.

ex:

Gpay
PhonePay
TempleRun
and etc.

Comments in java

Comments are created for documentation purpose.
Comments are used to improve readability of our code.
It is highly recommanded to use comments in our regular programming.
Comments will not display in output because our compiler does not compile the comments.
In java we have two types of comments.

1)Single Line Comment

It is used to comment a single line.

ex: // comment here

2)Multiple Line Comment

It is used to comment multiple lines.

```
ex:     /*
                -
                - comment here
                -
        */
```

ex:

```
//class declaration
class  Test
{
        //main method
        public static void main(String[] args)
        {
                //variable declaration
                int i=10;
                //output statement
                System.out.println(i);
        }
}
```

Q)Differences between python and java?

Python	Java
It is developed by Guido Van Rossum.	It is developed by James Gosling.
It is a product of Microsoft.	It is a product of Oracle Corporation.
It is a scripting language.	It is object oriented programming language.
It is a interpreted language.	It is a compiled language.
It contains PVM.	It contains JVM.
It is a dynamically typed language.	It is a statically typed language.
Performance is low.	Performance is high.
There is low security.	There is high security.
It contains less code.	It contains more code.

IT companies

We have two types of IT companies.

1)Service Based company

ex: TCS, Capgemini, Cognizent and etc.

2)Product Based company

ex: Microsoft, Oracle Corporation, IBM, Amazon and etc.

Naming Conventions in java

In java uppercase letters will consider as different and lowercase letters will consider as different.Hence we consider java is a case sensitive programming language.

As java is a case sensitive , we must and should follow naming conventions for following things.

ex: classes
interfaces
variables
methods
keywords
packages
constants

classes

In java, a class name must and should starts with uppercase letter and if it contains multiple words then each inner word starts with initcap.

ex: predefined classes	userdefined classes
System	Test
StringBuilder	DemoApp
FileWriter	ExampleApplication
BufferedReader	QualityThought
and etc.	and etc.

interfaces

In java, an interface name must and should starts with uppercase letter and if it contains multiple words then each inner word starts with initcap.

ex: predefined interfaces	userdefined interfaces
Serializable	ITest
Cloneable	IDemoApp
Enumeration	IExampleApplication
ListIterator	IQualityThought
and etc.	and etc.

variables

In java , a variable name must and should starts with lower case letter and if it contains multiple words then each inner word starts with initcap.

ex: predefined varibles	userdefined variables
out	empId
in	studName
err	deptNo
length	serialNo
and etc.	and etc.

methods

In java, a method name starts with lower case letter and if it contains multiple words then each inner word starts with initcap.

ex:	predefined methods	userdefined methods
	getClass()	calculateBill()
	setPriority()	getEmployeeDetails()
	setName()	setInfo()
	hashCode()	getStudentsDetails()
	toString()	and etc.
	and etc.	

keywords

In java, all keywords we need to write under lowercase letters only.

ex:	predefined keywords
	if
	else
	public
	static
	for
	void
	and etc.

packages

In java, all packages we need to write under lower case letters only.

ex:	predefined packages	userdefined packages
	java.lang	com.ihubtalent.www
	java.io	com.google.www
	java.util	com.facebook.www
	java.text	and etc.
	java.time	
	java.sql	
	and etc.	

constants

In java, all constants we need to write under uppercase letters only.

ex:	predefined constants	userdefined constants
	MAX_PRIORITY	LIMIT
	MIN_PRIORITY	DATA
	NORM_PRIORITY	and etc.
	MAX_VALUE	
	MIN_VALUE	
	and etc.	

Assignment

class : RajKumar

interface : IRajKumar

variable : rajKumar
method : rajKumar()
package : com.rajkumar.www
constant : RAJKUMAR or RAJ_KUMAR

Interview Questions

Q) Which is a default package in java?

java.lang package

Q) What is package?

Package is a collection of classes and interfaces.

Q) How many classes are there in java?

Java 7	-	4024 classes
Java 8	-	4240 classes
Java 9	-	6005 classes
Java 10	-	6002 classes

Q) What is Java?

Java is a object oriented, platform independent ,case sensitive, strongly typed checking , high level , open source programming language developed by James Gosling in the year of 1995.

Q)Differences between C++ and Java?

C++	Java
It is developed by Bjarne Stroustrup.	It is developed by James Gosling.
It is a partial object oriented programmin language.	It is purely object oriented programming language.
It is a platform dependent.	It is a platform independent.
Memory allocation and deallocation will taken create by a programmer.	Memory allocation and deallocation will taken care by a JVM.
It supports multiple inheritance.	It does not support multiple inheritance.
It supports pointers.	It does not support pointers.
It supports operator overloading.	It does not support operator overloading.
It supports preprocessor directory(#).	It does not support preprocessor directory(#).
It supports three access specifiers i.e public,private and protected.	It supports four access modifiers i.e default,public,private and protected.
It contains three loops i.e do while loop, while loop and for loop.	It contains four loops i.e do while loop, while loop, for loop and for each loop.

Q)What are the features of Java?
We have following important features in java.

1. Simple
2. Object oriented
3. Platform independent
4. Highly secured
5. Robust
6. Portable
7. Architecture Neutral
8. Multithreaded
9. Dynamic and etc.

Q)Who is the responsible to destroy objects in java?
Garbage Collector

Q)What is garbage collector ?
Garbage collector is used to destroy unused or useless objects from java.

Q)In how many ways we can call garage collector?
There are two ways to call garbage collector in java.

1) System.gc()
2) Runtime.getRuntime().gc()

History of Java

In 1990, Sun Micro system took one project to develop a software called consumer electronic device which can be controlled by a remote like setup box. That time project was called Stealth project and later is renamed to Green project.

James Gosling , Mike Sheridan, Patrick Naughton were there to develop this project and they met in a place called Aspan/Colarado to start the work with Graphic System.James Gosling decided to use C and C++ languages to develop this project.But the problem what they have faced is they are system dependent.Then James Gosling decided why don't we create our own programming language which is system independent.

In 1991, they have developed on programming language called an OAK.OAK means strength, itself is a coffee seed name and it is a national tree for many contries like Germany , France , USA and etc. Later in 1995 they have renamed OAK to Java.Java is a island of an Indonasia where first coffee of seed was produced and during the development of project they were consuming lot of coffee's.Hence symbol of java is a cup of coffee with saucer.

Interview questions
Q) Who is the creator of java?
James Gosling

Q) In which year java was developed?
In 1995

Q) Java originally known as ___ ?
OAK

Q) Java is a platform dependent or platform independent?
It is platform independent

Q) Java is a product of which company?
Oracle Corporation

Q) Which is the latest version of Java?
Java 20

Q)What is the difference between JDK , JRE or JVM ?

JDK

JDK stands for Java Development Kit.
It is a installable software which consist Java Runtime Environment (JRE), Java Virtual Machine (JVM), Compiler (javac) , Interpreter (java), archiever (.jar) , document generator (javadoc) and other tools needed for java application development.

JRE

JRE stands for Java Runtime Environment.
It provides very good environment to run java applications only.

JVM

JVM stands for Java Virtual Machine.
It is used to execute our program line by line procedure.

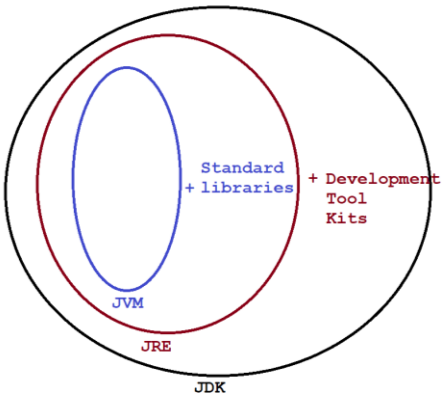


Diagram:java5.1

Identifiers

A name in java is called identifier.
It can be class name, variable name , method name or label name.
ex:

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

```
                int i = 10;
                System.out.println(i);
        }
}
```

Here Test,main,args and i are identifiers.

Rules to declare an identifiers

Rule1: Identifier will accept following characters.
ex: A-Z, a-z, 0-9, _, $

Rule2: If we take other characters then we will get compile time error.
ex:
```
int  $=10; //valid
int  _abcd; //valid
int  #=20; //invalid
```

Rule3: Identifier must and should starts with alphabet,underscore or dollar symbol but not with digits.
ex:
```
int  a1234; //valid
int  _1234; //valid
int  1abcd; //invalid
```

Rule4: Every identifier is a case sensitive.
ex:
```
int number;
int NUMBER;
int NumBer;
```

Rule5: We can't take reserved words as an identifier name.
ex:
```
int  if;//invalid
int  else; //invalid
```

Rule6: There is no length limit for an identifier but it is not recommanded to take more then 15 characters.

Rule7: We can take predefined classes and interfaces as an identifier name but it is not good programming practice.
ex:
```
int  System=10;
int  Runnable=20;
```

Reserved words

There are some identifiers which are reserved to associate some funcationality or meaning such type of identifiers are called reserved words.

Java supports 53 reserved words and it is divided into two types.

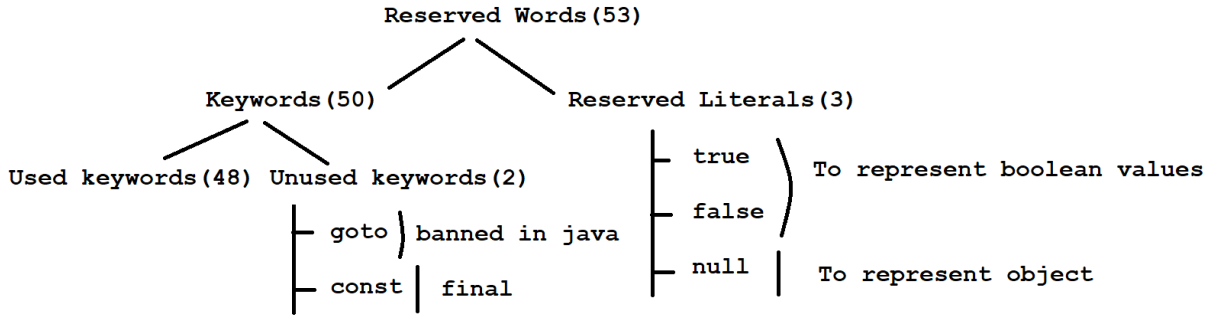


Diagram: java6.1

Used keywords with respect to class
package
import
interface
enum
class
implements
extends
Used keywords with respect to object
new
instanceof
this
super
Used keywords with respect to datatype
byte
short
int
long
float
double
boolean
char
Used keywords with respect to modifiers
default
public
private
protected
final
static
abstract
native
transient
volatile
synchronized
strictfp
Used keywords with respect to flow control
if
else
switch
case
break
continue
do
while
for

Used keywords with respect to returntype
void
Used keywords with respect exception handling
try
catch
throw
throws
finally
assert

Java

Version	:	Java 8
JDK	:	1.8v
Creator	:	James Gosling
Vendor	:	Oracle Corporation (Sun Micro System)
Open source	:	Open source
website	:	www.oracle.com/in/java
Tutorials	:	www.javatpoint.com www.javaus.com www.dzone.com www.w3school.com and etc.

Steps to setup Java environmental variables

step1: Make sure JDK 1.8 version installed successfully.
step2: Copy "lib" directory from java_home folder.
ex: C:\Program Files\Java\jdk1.8.0_181\lib
step3: Paste "lib" directory in Environmental variables.
ex:
Right click to MyPC --> properties --> Advanced system settings
Environmental variables -->

User variables --> click to new button -->
variable Name : CLASSPATH
variable value : C:\Program Files\Java\jdk1.8.0_181\lib;

System variables --> click to new button -->

variable Name : path
variable value : C:\Program Files\Java\jdk1.8.0_181\bin;

--> ok ---> ok --->ok.
step4: Check the environmental setup done perfectly or not.
ex: cmd> javap
cmd> java -version

Steps to develop first java application

step1: Make sure JDK 1.8 installed successfully.
step2: Make sure Environment setup done perfectly.
step3: Create a "javaprog" folder inside 'E' drive.
step4: Open the nodepad and develop simple Hello World program.

```
ex:     class  Test
        {
                public static void main(String[] args)
                {
                System.out.println("Hello World");
                }
        }
```

step5: Save above java program with same name as class name inside javaprog folder.
step6: Open the command prompt from javaprog folder or location.
step7: Compile the java program by using below command.

```
ex:     javaprog> javac   Test.java
                            |
                          filename
```

step8: Run the java program by using below command.

```
ex:     javaprog>  java    Test
                            |
                        classname
```

Interview Questions

Q)What is package?
A package is a collection of classes and interfaces.

Q)Which package is a default package in java?
java.lang package is a default package in java.

Q)What is Program?
A program is a set of instructions and instruction is a set of tokens.

Q)What is application?
Application is a collection of programs.

Q)What is software?
Software is a collection of applications.

Datatypes

Datatype describes what type of value we want to store inside a variable.
Datatype also tells how much memory has to be created for a variable.
In java datatypes are divided into two types.

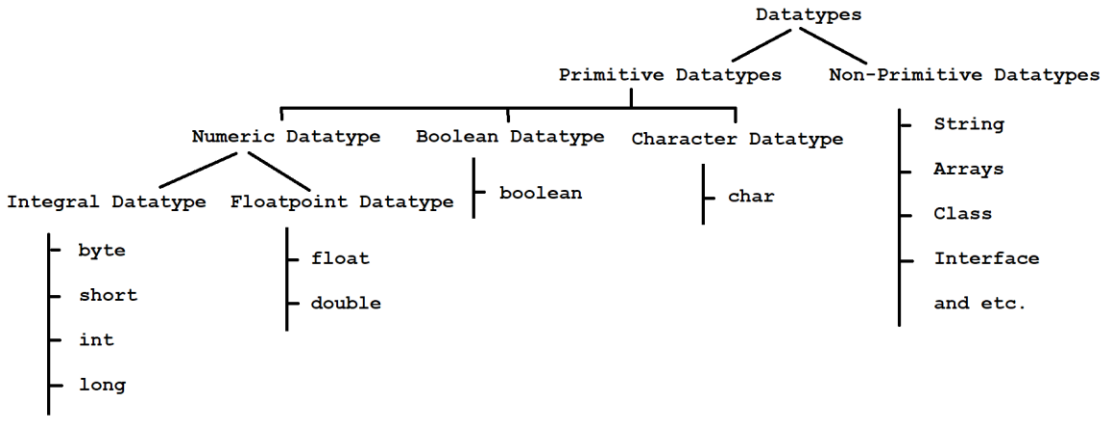


Diagram: java7.1

byte

It is a smallest datatype in java.
Size: 1 byte (8 bits)
Range: -128 to 127 (-2^7 to 2^7-1)
ex: 1) byte b=10;
System.out.println(b); // 10

2) byte b=130;
System.out.println(b); // C.T.E

3) byte b=10.5;
System.out.println(b); // C.T.E

short

It is rarely used datatype in java.
Size: 2 bytes (16 bits)
Range : -32768 to 32767 (-2^15 to 2^15-1)
ex: 1) byte b=100;
short s=b;
System.out.println(s); // 100

2) short s="hi";
System.out.println(s); // C.T.E

3) short s=true;
System.out.println(s); // C.T.E

int

It is mostly used datatype in java.
Size: 4 bytes (32 bits)
Range: -2147483648 to 2147483647 (-2^31 to 2^31-1)
ex: 1) int i=10.5;
System.out.println(i); // C.T.E

2) int i="false";
System.out.println(i); // C.T.E

3) int i=true;
System.out.println(i); // C.T.E

4) int i='a';
System.out.println(i); // 97

Note: In java for every character we have universal unicode value.
ex: A ---- 65
a ---- 97

long

If int datatype is not enough to hold large value then we need to use long datatype.
Sze: 8 bytes (64 bits)
Range: (-2^63 to 2^63-1)
ex: 1) long l=10.5;
System.out.println(l); // C.T.E

2) long l=true;
System.out.println(l); // C.T.E

3) long l="hi";
System.out.println(l); // C.T.E

4) long l=10;
System.out.println(l); //10

5) long l='A';
System.out.println(l); // 65

float

If we depend upon 4 to 6 decimal point of accuracy then we need to float.

Size: 4 bytes (32 bits)
Range: -3.4e38 to 3.4e38
To represent float value we need to suffix with 'f'.
ex:
10.5f

double

If we depend upon 14 to 16 decimal point of accuracy then we need to use double.
Size: 8 bytes (64 bits)
Range: -1.7e308 to 1.7e308
To represent double value we need to suffix with 'd'.
ex:
10.5d

ex:
1) float f=10.5f;
System.out.println(f); // 10.5

2) float f=10;
System.out.println(f); // 10.0

3) float f='a';
System.out.println(f); //97.0

ex:

1) double d=10.5d;
System.out.println(d); // 10.5

2) double d=10;
System.out.println(d); // 10.0

3) double d='A';
System.out.println(d); //65.0

boolean

It is used to represent boolean values either true or false.
Size : (Not Applicable - 1 bit)
Range : (Not Applicable)
ex:

1) boolean b="false";
System.out.println(b); // C.T.E

2) boolean b=TRUE;
System.out.println(b); // C.T.E

3) boolean b=true;
System.out.println(b) // true

char

It will take single character which is enclosed in a single quotation.
Size : 2 bytes (16 bits)
Range : 0 to 65535
ex:

1) char c='a';
System.out.println(c); // a

2) char c=65;
System.out.println(c); // A

3) char c="a";
System.out.println(c); // C.T.E

Datatype	Size	Range	Wrapper class	Default value
byte	1 byte	-128 to 127	Byte	0
short	2 bytes	-32768 to 32767	Short	0
int	4 bytes	-2147483648 to 2147483647	Integer	0
long	8 bytes	-2^63 to 2^63-1	Long	01
float	4 bytes	-3.4e38 to 3.4e38	Float	0.0
double	8 bytes	-1.7e308 to 1.7e308	Double	0.0
boolean	-	-	Boolean	false
char	2 bytes	0 to 65535	Character	0(space)

Diagram: java7.2

Q)Write a java console to display the range of byte datatype?

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(Byte.MIN_VALUE);
		System.out.println(Byte.MAX_VALUE);
	}
}
```

Q)Write a java console to display the range of int datatype?

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(Integer.MIN_VALUE);
		System.out.println(Integer.MAX_VALUE);
	}
}
```

Q)Is java purely object oriented or not?

No, Java will not consider as purely object oriented programming language because it does not support many OOPS concepts like multiple inheritance, operator overloading and more ever we depends upon primitive datatypes which are non-objects.

Internal Architecture of JVM

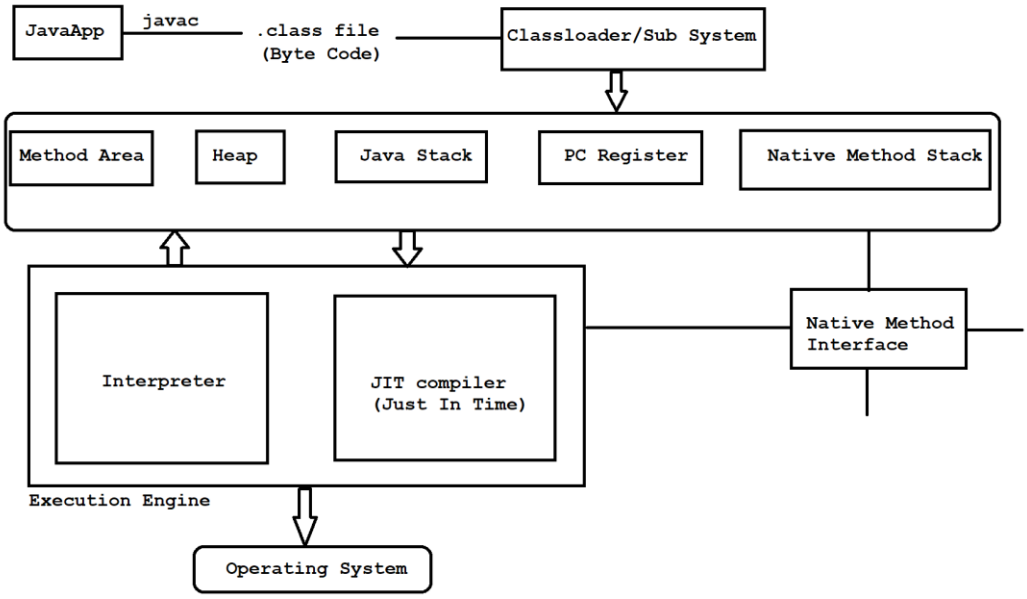


Diagram: java8.1

Our java application contains java code instructions.Once if we compile ,java code instructions convert to byte code instructions in .class file.
JVM will invoke a module called classloader or subsystem to load all the byte code instructions from .class file. The work of classloader is to check byte code instructions are proper or not.If it is not proper then it will refuse the execution.If it is proper then it will allocate the memories.

We have five types of memories.
1)Method Area
Method area contains code of a variable, code of a method and code of a class.

2)Heap
Our object creation will store in heap area.

Note: Whenever JVM loads byte code instructions from .class file it will create method area and heap area automatically.

3)Java Stack
Java methods will store in method area but to execute those methods we required some memory.That memory will be allocated in java stack.

4)PC register
It is a program counter register which is used to track the address of an instructions.

5)Native Method Stack
Java methods will execute in method area.Similary native methods will execute in native method stack.
But we can't execute native methods directly we required a program called Native method interface.

Execution Engine

Execution engine contains interpreter and JIT compiler.
Whenever JVM loads byte code instructions from .class file it will use interpreter and JIT compiler simultenously.
Interpreter is used to execute our program line by line procedure.
JIT compiler is used to increase the execution speed of our program.

Interview Questions

Q)A .class file contains what code ?
Byte code

Q)How many memories are there in java?
There are five memories in java.
1) Method area
2) Heap
3) Java Stack
4) PC register
5) Native method stack

Q)What is JIT compiler ?
It is a part of a JVM which is used to increase the execution speed of our program.

Q)How many classloaders are there in java?
We have three predefined classloaders in java.
1)Bootstrap classloader (It loads rt.jar file)
2)Extension classloader (It loads all the jar file from ext folder)
3)Application/System classloader (It loads .class file from CLASSPATH)

Types of variables

A name which is given to a memory location is called variable.
Purpose of variable is used to store the data.
In java, We have two types of variables.

1)Primitive variables
It is used to represent primitive values.
2)Reference variables
It is used to represent object reference.
ex: Student s=new Student();
|
reference variable

Based on the position and execution these variables are divided into three types.
1)Instance variables / Non-static variables
2)Static variables / Global variables
3)Local variables / Temperory variables / Automatic variables

1)Instance variables

A value of a variable which is varied(changes) from object to object is called instance variable.

Instance variable will be created at the time of object creation and it will destroy at the time of object destruction.Hence scope of instance variable is same as scope of an object.

Instance variable store in Heap area as a part of an object.

Instance variable must and should declare immediately after the class but not inside methods, blocks and constructors.

Instance variable we can access directly from instance area but we can't access directly from static area.

To access instance variable from static area we need to create object reference.

ex:1

```
class  Test
{
        //instance variable
        int  i = 10;
        public static void main(String[] args)
        {
                System.out.println(i);//C.T.E
        }
}
```

ex:2

```
class  Test
{
        //instance variable
        int  i = 10;
        public static void main(String[] args)
        {
                Test t=new Test();
                System.out.println(t.i);//10
        }
}
```

Note: If we won't initialize any value to instance variable then JVM will initialized default values.

ex:3

```
class  Test
{
        //instance variable
        boolean b;
        public static void main(String[] args)
        {
                Test t=new Test();
                System.out.println(t.b);//false
        }
}
```

ex:4

```
class  Test
{
        public static void main(String[] args)
        {
                //calling
                Test t=new Test();
                t.m1();
        }
        //non-static method
        public void m1()
        {
                System.out.println("instance-method");
        }

}
```

2)Static variables

A value of a variable which is not varied from object to object is called static variable.

A static variable will be created at the time of class loading and it will destroy at the time of class unloading .Hence scope of static variable is same as scope of .class file.

Static variable will store in method area.

Static variable must and should declare immediately after the class by using static keyword but not inside methdods,blocks and constructors.

Static variable can access directly from instance area and static area.

Static varible can access by using object reference and classname.

ex:1

```
class  Test
{
        //static variable
        static int i=10;
        public static void main(String[] args)
        {
                System.out.println(i);//10
                Test t=new Test();
                System.out.println(t.i); //10
                System.out.println(Test.i);//10
        }

}
```

Note: If we won't initialize any value to static variable then JVM will initialized default values.

ex:2

```
class  Test
{
        //static variable
```

```
	static String s;
	public static void main(String[] args)
	{
		System.out.println(s);//null
	}
}
```

ex:3

```
class  Test
{
	public static void main(String[] args)
	{
		//calling
		m1();
		Test t=new Test();
		t.m1();
		Test.m1();
	}
	//static method
	public static void m1()
	{
		System.out.println("static-method");
	}
}
```

3)Local variables

To meet temperory requirement, programmers will declare some variables inside methods , blocks and constructors such type of variables are called local variables.

A local variable will be created at the time of execution block and it will destroy when execution block is executed.Hence scope of local variable is same as scope of execution block where it is declared.

Local variable will store in java stack memory.

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		//local variable
		int i=10;
		System.out.println(i);//10
	}

}
```

Note: IF we won't initialize any value to local variable then JVM will not initialized any default value.

ex:2

```
class  Test
{
        public static void main(String[] args)
        {
                //local variable
                int i;
                System.out.println(i);//C.T.E
        }

}
```

o/p: variable i might not have been initialized

Main method

Our program contains main method or not.
Either it is properly declared or not.
It is not a responsibility of a compiler to check.
It is a liability of a JVM to look for main method always at runtime.
JVM always look for main method with following signature.
syntax: public static void main(String[] args)
If we perform any changes in above signature then JVM will throw one runtime error called main method not found.

public	JVM wants to call main method from anywhere.
static	JVM wants to call main method without using object reference.
void	Main method does not return anything to JVM.
main	It is a identifier given to a main method.
String[] args	It is a command line argument.

We can perform following changes in main method.
1)Order of modifiers is not important , incase of public static we can declare static public also.
ex: static public void main(String[] args)
2) We change String[] in following acceptable formats.
ex: public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
3) We can replace String[] with var-arg parameter.
ex: public static void main(String... args)
4) We can change args with any java valid identifier.
ex: public static void main(String[] ihub)
5) Main method will accept following modifiers.
ex: synchronized
strictfp
final

Command Line arguments

Arguments which are passing through command prompt such type of arguments are called command line arguments.

In command line arguments we need to pass out input values always at runtime.

```
ex:     javac   Test.java
        java    Test  101  raja  M  1000.0
                      |   |     |      |____args[3]
                      |   |     |___________args[2]
                      |   |_________________args[1]
                      |_____________________args[0]
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println(args[0]);
                System.out.println(args[1]);
                System.out.println(args[2]);
                System.out.println(args[3]);
        }
}
```

System.out.println()

It is a output statement in java.

Whenever we want to display any data or userdefined statements then we need to use output statement.

```
syntax:         static variable
                |
        System.out.println(" ");
        |               |
        predefined      predefined method
        final class
```

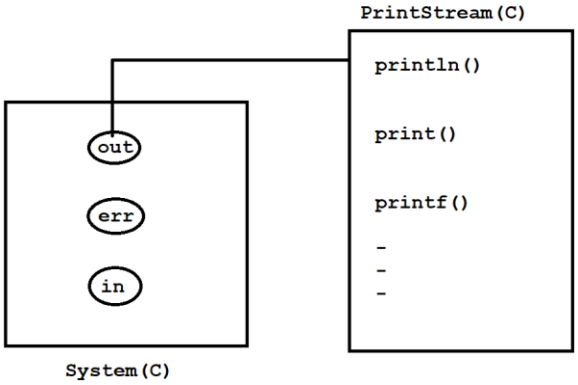


Diagram: java9.1

Various ways to display the data

1) System.out.println("Hello World");
2) int i=10;
   System.out.println(i);
   System.out.println("The value is ="+i);
3) int i=10,j=20;

```
        System.out.println(i+" "+j);
        System.out.println(i+" and "+j);
4)      int i=1,j=2,k=3;
        System.out.println(i+" "+j+" "+k);
```

Q)What is the difference b/w System.out.println() and System.err.println()?

System.out.println()

It is used to display the output only one console.

ex:

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("Hello World");
        }
}
```

System.err.println()

It is used to display the output on console as well it will redirect to some physical file.

ex:

```
class Test
{
        public static void main(String[] args)
        {
                System.err.println("Hello World");
        }
}
o/p:    javac   Test.java
        java    Test 2>abc.txt
```

Fully Qualified Name

Fully qualified name means we will declare our class or interface along with package name.

It is used to improve the readability of our code.

ex:

```
class Test
{
        public static void main(String[] args)
        {
                java.util.Date d=new java.util.Date();
                System.out.println(d);
        }
}
```

Import Statements

Whenever we use import statement we should not fully qualified name.

Using short name also we can achieve.

We have three types of import statements in java.
1)Explicit class import
2)Implicit class import
3)static import

1)Explicit class import

This type of import statement is higly recommanded to use because it will improve readability of our code.
ex:

```
import java.time.LocalDate;
import java.time.LocalTime;
class Test
{
	public static void main(String[] args)
	{
		LocalDate date=LocalDate.now();
		System.out.println(date);
		LocalTime time=LocalTime.now();
		System.out.println(time);
	}
}
```

2)Implicit class import

This type of import statement is not recommanded to use because it will reduce the readability of our code.
ex:

```
import java.time.*;
class Test
{
	public static void main(String[] args)
	{
		LocalDate date=LocalDate.now();
		System.out.println(date);
		LocalTime time=LocalTime.now();
		System.out.println(time);
	}
}
```

3)static import

Using static import we can call static members directly.
Often use of static makes our program unreadable or complex.
ex:

```
import static java.lang.System.*;
class Test
{
	public static void main(String[] args)
	{
		out.println("stmt1");
		out.println("stmt2");
```

```
                out.println("stmt3");
        }
}
```

ex:

```
import static java.lang.System.*;
class Test
{
        public static void main(String[] args)
        {
                out.println("stmt1");
                exit(0);
                out.println("stmt3");
        }
}
```

Basic Java Programs

Q) Write a java program to display sum of two numbers?

```
import java.util.Scanner;
class Example1
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the First Number :");
                int a=sc.nextInt();
                System.out.println("Enter the Second Number :");
                int b=sc.nextInt();
                //logic
                int c=a+b;
                System.out.println("sum of two numbers is ="+c);
        }
}
```

Q)Write a java console to perform sum of two numbers without using third variable?

```
import java.util.Scanner;
class Example2
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the First Number :");
                int a=sc.nextInt();
                System.out.println("Enter the Second Number :");
                int b=sc.nextInt();
                System.out.println("sum of two numbers is ="+(a+b));
```

```
	}
}
```

Q)Write a java program to find out square of a given number?

	input: 5
	output: 25

```
import java.util.Scanner;
class Example3
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the Number :");
		int n=sc.nextInt();
		int square=n*n;
		System.out.println("square of a given number is ="+square);
	}
}
```

Q)Write a java program to find out cube of a given number?

input: 5
output:125
ex:

```
import java.util.Scanner;
class Example4
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the Number :");
		int n=sc.nextInt();
		int cube=n*n*n;
		System.out.println("Cube of a given number is ="+cube);
	}
}
```

Q)Write a java program to find out area of a circle?

```
import java.util.Scanner;
class Example5
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the radius :");
```

```
		int r=sc.nextInt();
		//logic
		float area=3.14f*r*r;
		System.out.println("Area of a circle is ="+area);
	}
}
```

Q)Write a java program to find out perimeter of a circle?

```
import java.util.Scanner;
class Example6
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the radius :");
		int r=sc.nextInt();
		//logic
		float perimeter=2*3.14f*r;
		System.out.println("Perimeter of a circle is ="+perimeter);
	}
}
```

Q)Write a java program to perform swapping of two numbers?

	input: a = 10 and b = 20
	output: a = 20 and b = 10

```
import java.util.Scanner;
class Example7
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the First Number :");
		int a=sc.nextInt();
		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();

		System.out.println("Before swapping a="+a+" and b="+b);
		//logic
		int temp=a;
		a=b;
		b=temp;
		System.out.println("After swapping a="+a+" and b="+b);

	}
}
```

Q)Write a java program to perform swapping of two numbers without using third variable?

```
import java.util.Scanner;
class Example8
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the First Number :");
		int a=sc.nextInt();
		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();
		System.out.println("Before swapping a="+a+" and b="+b);
		//logic
		a=a+b;
		b=a-b;
		a=a-b;
		System.out.println("After swapping a="+a+" and b="+b);

	}
}
```

Q)What is a java program to accept salary and find out 10% of TDS ?

```
import java.util.Scanner;
class Example9
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the salary :");
		int salary=sc.nextInt();
		//logic
		float tds=(float)salary*10/100;
		System.out.println("10 percent of TDS is ="+tds);

	}
}
```

Q)Write a java program to convert CGPA to percentage?

```
import java.util.Scanner;
class Example10
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the CGPA  :");
```

```
		float cgpa=sc.nextFloat();

		//logic
		float percentage=cgpa*9.5f;
		System.out.println("CGPA to percentage is ="+percentage);
	}
}
```

Assignment
Q) Write a java program to find out area of a rectangle?
Q) Write a java program to find out area of a triangle?

Typecasting
The process of converting from one datatype to another datatype is called typecasting.
In java, typecasting can be performed in two ways.
1)Implicit typecasting
2)Explicit typecasting

1)Implicit typecasting
If we want to store small value into a bigger variable then we need to use implicit typecasting.
A compiler is responsible to perform implicit typecasting.
There is no possibility to loss the information.
It is also known as widening or upcasting.
We can perform implicit typecasting as follow.

```
ex:	byte	--->	short
				--->
					int  -->long -->float -->double
				--->
			char
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		char  ch='a';

		long l=ch;

		System.out.println(l);//97
	}
}
```

ex:

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

```
	{
		int i=10;

		double d=i;

		System.out.println(d); // 10.0
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		byte b=10;

		int i=b;

		System.out.println(i); // 10
	}
}
```

2)Explicit typecasting

If we want store big value into a smaller variable then we need to use explicit typecasting.
A programmer is responsible to perform explicit typecasting.
There is a possibility to loss the information.
It is also known as Narrowing or Downcasting.
We can perform explicit typecasting as follow.

```
ex:	byte	<---	short
				<---
					int  <-- long <--float <--double
				<---
			char
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=65;

		char ch=(char)i;

		System.out.println(ch);//A
	}
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                float f=10.5f;

                int i=(int)f;

                System.out.println(i);//10
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=130;

                byte b=(byte)i;

                System.out.println(b); // -126
        }
}
```

Java Source File Structure

case1:

A java program can have multiple classes.

If a java program contains multiple classes then we need to checkwhich class contains main method , that class will consider as main class.

ex: B.java

```
class A
{
        -
}
class B
{
        public static void main(String[] args)
        {
                -
        }
}
```

If we compile above program we will get two .class files i.e

A.class and B.class.

case2:

If a java program contains multiple classes with main method then we need to declare one class as public and that class will consider as main class.

ex:

A.java

```
-------
public class A
{
        public static void main(String[] args)
        {
                System.out.println("A class");
        }
}
class B
{
        public static void main(String[] args)
        {
                System.out.println("B class");
        }
}
class C
{
        public static void main(String[] args)
        {
                System.out.println("C class");
        }
}
```

> javac A.java (A.class,B.class and C.class)
> java A (A class will execute)
> java B (B class will execute)
> java C (C class will execute)

Types of Blocks in java

A block is a set of statements which is enclosed in a curcly braces i.e {}.

syntax:

```
//block
{
        -
        - //set of statements
        -
}
```

We have three types of blocks in java.

1)Instance block
2)Static block
3)Local block

1)Instance block

Instance block is used to initialize the instance variable.
Instane block will execute when we create an object.
We can declare instance block as follow.
syntax:

```
        //instance block
        {
                -
                - //set of statements
                -
        }
```

Instance block must and should declare immediately after the class but not inside methods and constructors.
ex:1

```
class Test
{
        //instance block
        {
                System.out.println("instance-block");
        }

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

o/p: main-method

ex:

```
class Test
{
        //instance block
        {
                System.out.println("instance-block");
        }

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

o/p: main-method
instance-block

ex:

```
class Test
{
	//instance block
	{
		System.out.println("instance-block");
	}

	public static void main(String[] args)
	{
		Test t1=new Test();
		System.out.println("main-method");
		Test t2=new Test();
	}
}
```

o/p:
```
instance-block
main-method
instance-block
```

ex:

```
class Test
{
	//instance variable
	int i;

	//instance block
	{
		i=100;
	}

	public static void main(String[] args)
	{
		Test t=new Test();
		System.out.println(t.i);//100
	}
}
```

2)Static block

Static block is used to initialize the static variables.
Static block will execute at the time of classloading.
We can declare static block as follow.
syntax:

```
	//static block
	static
	{
```

```
        -
        - //set of statements
        -
    }
```

Static block must and should declare immediate after the class using static keyword but not inside methods and constructors.

ex:1

```
class Test
{
    //static block
    static
    {
        System.out.println("static-block");
    }

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

o/p: static-block
main-method

ex:2

```
class Test
{
    //static block
    static
    {
        System.out.println("static-block");
    }

    //instance block
    {
        System.out.println("instance-block");
    }

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

o/p: static-block
instance-block
main-method

ex:3

```
class Test
{
	//static variable
	static int i;

	//static block
	static
	{
		i=200;
	}

	public static void main(String[] args)
	{
		System.out.println(i);//200
	}
}
```

3)Local block

A local block is used to intialize the local variables.
Local will execute just like normal statement.
We can declare local block as follow.
syntax:

```
	//local block
	{
		-
		- //set of statements
		-
	}
```

Local block must and should declare inside methods and constructors.
ex:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		//local block
		{
			System.out.println("stmt2");
		}
		System.out.println("stmt3");
	}
}
```

o/p:
```
	stmt1
	stmt2
	stmt3
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		//local variable
		int i;

		//local block
		{
			i=300;
		}

		System.out.println(i); // 300
	}
}
```

Interview Question

Q)Can we execute java program without main method ?

Yes , Till 1.6 version it is possible to execute java program without main method by using static block.But from 1.7 version onwards it is not possible to execute java program without main method.

ex:

```
class Test
{
	static
	{
		System.out.println("Hello World");
		System.exit(0);
	}
}
```

Operators

Operator is a symbol which is used to perform some operations on operands.

ex: a + b

Here + is a operator

Here a and b are operands

It can be arithmetic operation, logical operation, bitwise operation , conditional operation and etc.

We have following list of operators in java.

1)Assignment operators
2)Ternary operators/Conditional operators
3)Logical operators
4)Bitwise operators
5)Arithmetic operators
6)Relational operators

7)Unary operators

1)Assignment operators
ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		i=20;
		i=30;
		System.out.println(i); // 30
	}
}
```

Note: Reinitialization is possible in java.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		final int i=10;
		i=20;
		i=30;
		System.out.println(i); // C.T.E
	}
}
```

Note: We can't assign the values to final variable

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=1,2,3,4,5;

		System.out.println(i); //C.T.E
	}
}
```

Note: Illegal start of expression

ex:

```
class Test
{
	//global variable
```

```
	static int i=100;

	public static void main(String[] args)
	{
		//local variable
		int i=200;

		System.out.println(i); //200
	}
}
```
Note: Here priority goes to local variable.

ex:
```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(10%2); // 0
		System.out.println(10%20); // 10
		System.out.println(40%3); // 1
		System.out.println(40%80); // 40
	}
}
```

ex:
```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(10/2); // 5
		System.out.println(10/20); // 0
		System.out.println(40/3); // 13
		System.out.println(40/80); // 0
	}
}
```

2)Ternary operator / Conditional operator

	syntax:	(condition)?value1:value2;

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b=(5>2)?true:false;
		System.out.println(b);//true
```

```
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b=(5>20)?true:false;
		System.out.println(b);//false
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=(4>2)?1:0;

		System.out.println(i);//1
	}
}
```

Q)Write a java program to find out greatest of two numbers?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the First Number :");
		int a=sc.nextInt();

		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();

		//logic
		int max=(a>b)?a:b;

		System.out.println("Greatest of two numbers is ="+max);
```

```
	}
}
```

Q)Write a java program to find out greatest of three numbers?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the First Number :");
		int a=sc.nextInt();

		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();

		System.out.println("Enter the Third Number :");
		int c=sc.nextInt();

		//logic
		int max=(a>b)?((a>c)?a:c):((b>c)?b:c);

		System.out.println("Greatest of three numbers is ="+max);
	}
}
```

3)Logical operators
logical AND operator(&&)
Truth table

T	T	= T
T	F	= F
F	T	= F
F	F	= F

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b=((5>6) && (9<10))?true:false;
		System.out.println(b);//false
	}
```

```
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b=((5>2) && (9<10))?true:false;
		System.out.println(b);//true
	}
}
```

logical OR operator(||)

Truth table

T	T	= T
T	F	= T
F	T	= T
F	F	= F

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b=(5>2) || (6<2);

		System.out.println(b);//true
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b= (5>7) || false;
		System.out.println(b);//false
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b= (5>20) && (9<10) || false;

		System.out.println(b);//false
	}
}
```

Logical NOT operator (!)
ex:
```
class Test
{
	public static void main(String[] args)
	{
		boolean b=!(5>2);

		System.out.println(b);//false
	}
}
```

ex:
```
class Test
{
	public static void main(String[] args)
	{
		boolean b=!(6>90);

		System.out.println(b);//true
	}
}
```

How to convert decimal number to binary number
decimal number : 10
binary number : 1010

```
		2|10--- 0
		2|5--- 1
		2|2--- 0          ^
		  1               |
		--------------------
		1010
```

How to convert binary number to decimal number
binary number : 1010
decimal number : 10

```
1010
  <---
0*1 + 1*2 +  0*4 + 1*8
0  + 2 + 0 + 8
10
```

4) Bitwise Operators
Bitwise AND operator(&)
Bitwise AND operator deals with binary numbers.
Truth table

T	T	= T
T	F	= F
F	T	= F
F	F	= F

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int a=10,b=15;
		int c = a & b;
		System.out.println(c);//10
	}
}
/*
	10 - 1010
	15 - 1111
	   ----------
	&  - 1010
	                  <--

	0*1 + 1*2 + 0*4 + 1*8

	0 + 2 + 0 + 8 = 10
*/
```

ex:

```
class Test
{
```

```
	public static void main(String[] args)
	{
		int a=10,b=5;
		int c = a & b;
		System.out.println(c);//0
	}
}
/*
	10 - 1010
	5  - 0101
	   ----------
	&  - 0000
*/
```

Bitwise OR operator(|)
Bitwise OR operator deals with binary numbers.
Truth table

T	T	= T
T	F	= T
F	T	= T
F	F	= F

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int a=10,b=15;
		int c = a | b;
		System.out.println(c);//15
	}
}
/*
	10 - 1010
	15 - 1111
	   ----------
	|  - 1111
*/
```

Bitwise XOR operator(^)
Bitwise XOR operator deals with binary numbers.
Truth table

T	T	= F
T	F	= T
F	T	= T
F	F	= F

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int a=10,b=15;
		int c = a ^ b;
		System.out.println(c);//5
	}
}
/*
	10 - 1010
	15 - 1111
	   ----------
	^  - 0101
*/
```

Bitwise NOT operator (~)

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = ~10;

		System.out.println(i); // -11
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = ~56;

		System.out.println(i); // -57
	}
}
```

ex:

```
class Test
{
```

```
	public static void main(String[] args)
	{
		int i = ~(-35);

		System.out.println(i); // 34
	}
}
```

5)Arithmetic Operators
% - modules
/ - division
* - multiplication
+ - addition
- - subtraction

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i= 8+6%10+6/3+5*2+7/10+9-12;

		System.out.println(i); //23
	}
}
/*
	8 + 6%10 + 6/3 + 5*2 + 7/10  + 9-12

	8 + 6 + 2 + 10 + 0 + 9 - 12

   35 - 12

	23
*/
```

6)Relational operators

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(5>10); // false
		System.out.println(5>=10); // false

		System.out.println(5<10); // true
		System.out.println(5<=5); // true
```

```
                System.out.println(10 == 10); // true
                System.out.println(10 == 20); // false
                System.out.println(10 != 10); // false
                System.out.println(10 != 20); // true
        }
}
```

Right Shift operator (>>)
10 >> 1 = 10 / 2
10 >> 2 = 10 / 4
10 >> 3 = 10 / 8
10 >> 4 = 10 / 16
10 >> 5 = 10 / 32

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i = 10 >> 3;

                System.out.println(i); // 10 / 8 = 1
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i = 100  >> 6;

                System.out.println(i); // 100 / 64  = 1
        }
}
```

Left Shift operator (<<)
10 << 1 = 10 * 2
10 << 2 = 10 * 4
10 << 3 = 10 * 8
10 << 4 = 10 * 16
10 << 5 = 10 * 32

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = 10 << 4;

		System.out.println(i); // 10 * 16 = 160
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = 100 << 2;

		System.out.println(i); // 100 * 4 = 400
	}
}
```

Increment/Decrement Operators (++/--)
We have two increment operators.
i) post increment
	ex:		i++
ii)pre increment
	ex:		++i

We have two decrement operators.
i) post decrement
	ex:		i--
ii)pre decrement
	ex:		--i
Post increment/decrement
Rule1: First Take
Rule2: Then Change

ex:

```
class Test
{
	public static void main(String[] args)
	{
			int i=10;
			i++;
```

```
                System.out.println(i);//11
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=10;

                System.out.println(i++);//10
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=10;

                int j=i++;

                System.out.println(i+" "+j);//11 10
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=10;

                int j=i-- + i--; // 10 + 9

                System.out.println(i+" "+j);// 8  19
        }
}
```

ex:

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

```
                int i=10;

                int j=i++ + i++ + i++; // 10 + 11  + 12

                System.out.println(i+" "+j);//  13   33
        }
}
```

Pre increment/decrement
Rule1: First Change
Rule2: Then Take

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=10;

                ++i;

                System.out.println(i);//11
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=10;

                System.out.println(++i);//11
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int i=10;
                int j=++i;
                System.out.println(i+" "+j);//11 11
        }
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
			int i=10;

			System.out.println(i++ + ++i);//10 + 12 =22
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
			int i=100;

			100++;

			System.out.println(i);//C.T.E
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
			int i=10;

			System.out.println(++(i++));//C.T.E
	}
}
```

Control Statements

Control statement enables the programmer to control flow of the program.

Control statement allows us to make decisions, to jump from one section of code to another section and to execute the code repeatedly.

We have four control statements in java.

1)Decision making statement
2)Selection statement
3)Iteration statement
4)Jump statement

1)Decision making statement
It is used to create a conditions is our program.
Decision making statement is possible by using following ways.
i)if stmt
ii)if else stmt
iii)if else if ladder
iv) nested if stmt

i)if stmt
It will execute the source code only if our condition is true.
syntax:

```
if(condition/expression)
{
        -
        - //code to be execute
        -
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("stmt1");
                if(!(5>20))
                {
                        System.out.println("stmt2");
                }
                System.out.println("stmt3");
        }
}
```

o/p:

```
stmt1
stmt2
stmt3
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("stmt1");
                if(!(5>2))
                {
                        System.out.println("stmt2");
```

```
		}
		System.out.println("stmt3");
	}
}
```
o/p:

stmt1
stmt3

ex:

```
class Test
{
	public static void main(String[] args)
	{
		if((5>2) && (8<3))
			System.out.println("stmt1");
			System.out.println("stmt2");
			System.out.println("stmt3");
	}
}
```

Note:
Declaration of curly braces is optional.
If we won't defined curly braces then compiler will add curcly braces at the time of compilation only to first statement.

Q)Write a java program to find out greatest of two numbers?
```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the First Number :");
		int a=sc.nextInt();
		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();

		if(a>b)
			System.out.println(a+" is greatest");
		if(b>a)
			System.out.println(b+" is greatest");

	}
}
```

Q)Write a java program to find out greatest of three numbers?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the First Number :");
		int a=sc.nextInt();

		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();

		System.out.println("Enter the Third Number :");
		int c=sc.nextInt();

		if((a>b) && (a>c))
			System.out.println(a+" is greatest");
		if((b>a) && (b>c))
			System.out.println(b+" is greatest");
		if((c>a) && (c>b))
			System.out.println(c+" is greatest");

	}
}
```

ii)if else stmt

It will execute the source code either our condition is true or false.

syntax:

```
	if(condition/expression)
	{
		- //code to be execute if cond is true
	}
	else
	{
		- //code to be execute if cond is false
	}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
```

```
		if(true)
		{
			System.out.println("stmt2");
		}
		else
		{
			System.out.println("stmt3");
		}
		System.out.println("stmt4");
	}
}
```
o/p:

stmt1
stmt2
stmt4

ex:

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(false)
		{
			System.out.println("stmt2");
		}
		else
		{
			System.out.println("stmt3");
		}
		System.out.println("stmt4");
	}
}
```
o/p:

stmt1
stmt3
stmt4

Q)Write a java program to find out given age is eligible to vote or not?
```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
```

```
		System.out.println("Enter the age :");
		int age=sc.nextInt();

		if(age>=18)
			System.out.println("U r eligible to vote");
		else
			System.out.println("U r not eligible to vote");
	}
}
```

Q)Write a java program to check given number is even or odd?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		if(n%2==0)
			System.out.println("It is even number");
		else
			System.out.println("It is odd number ");
	}
}
```

Q)Write a java program to check given number is odd or not ?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		if(n%2==1 || n%2!=0)
			System.out.println("It is odd number");
		else
			System.out.println("It is not odd number ");
	}
}
```

Q)Write a java program to find out given year is a leap year or not?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the year :");
		int year=sc.nextInt();

		if(year%4==0)
			System.out.println("It is a leap year");
		else
			System.out.println("It is not a leap year ");
	}
}
```

Q)Write a java program to find out given number is +ve or -ve ?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		if(n==0)
		{
		System.out.println("It is not a positive or negative number");
			System.exit(0);
		}

		if(n>0)
			System.out.println("It is a positive number");
		else
			System.out.println("It is a negative number ");
	}
}
```

iii) if else if ladder

It will execute the source code based on multiple conditions.

syntax:

```
	if(cond1)
```

```
        {
                - //code to be execute if cond1 is true
        }
        else if(cond2)
        {
                - //code to be execute if cond2 is true
        }
        else if(cond3)
        {
                - //code to be execute if cond3 is true
        }
        else
        {
                - //code to be execute if all conditions are false.
        }
```

ex:

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter the option :");
                int opt=sc.nextInt();

                if(opt==100)
                        System.out.println("It is a police number");
                else if(opt==103)
                        System.out.println("It is a enquiry number");
                else if(opt==108)
                        System.out.println("It is a emergency number");
                else
                        System.out.println("Invalid option");
        }
}
```

Q)Write a java program to find out given alphabet is a vowel or not?

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
```

```
		System.out.println("Enter the alphabet :");
		char ch=sc.next().charAt(0);

		if(ch=='a')
			System.out.println("It is a vowel");
		else if(ch=='e')
			System.out.println("It is a vowel");
		else if(ch=='i')
			System.out.println("It is a vowel");
		else if(ch=='o')
			System.out.println("It is a vowel");
		else if(ch=='u')
			System.out.println("It is a vowel");
		else
			System.out.println("It is not a vowel");
	}
}
```

Q)Write a java program to find out given alphabet is a upper case letter, lower case letter, digit or a special symbol?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the alphabet :");
		char ch=sc.next().charAt(0);

		if(ch>='a' && ch<='z')
			System.out.println("It is a lowercase letter");
		else if(ch>='A' && ch<='Z')
			System.out.println("It is a uppercase letter");
		else if(ch>='0' && ch<='9')
			System.out.println("It is a Digit");
		else
			System.out.println("It is a Special symbol");
	}
}
```

iv)nested if stmt
If stmt contains another if stmt is called nested if stmt.
syntax:

```
	if(condition)
```

```
	{
		if(condition)
		{
			-
			- //code to be execute
			-
		}
	}
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(true)
		{
			System.out.println("stmt2");
			if(2>1)
			{
				System.out.println("stmt3");
			}
			System.out.println("stmt4");
		}
		System.out.println("stmt5");
	}
}
```

o/p:

```
	stmt1
	stmt2
	stmt3
	stmt4
	stmt5
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(false)
		{
			System.out.println("stmt2");
			if(2>1)
			{
```

```
                        System.out.println("stmt3");
                    }
                    System.out.println("stmt4");
                }
                System.out.println("stmt5");
        }
}
```

o/p:

```
        stmt1
        stmt5
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("stmt1");
                if(true)
                {
                        System.out.println("stmt2");
                        if(2>10)
                        {
                                System.out.println("stmt3");
                        }
                        System.out.println("stmt4");
                }
                System.out.println("stmt5");
        }
}
```

o/p:

```
        stmt1
        stmt2
        stmt4
        stmt5
```

Q)Write a java program to find out given number is +ve or -ve by using nested if stmt?

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter the number :");
                int n=sc.nextInt();
```

```
		if(n!=0)
		{
			if(n>0)
			{
				System.out.println("It is +ve number");
				System.exit(0);
			}
			System.out.println("It is -ve number");
		}
	}
}
```

2)Selection statement

Switch case

It will execute the source code based on multiple conditions.

It is similar to if else if ladder.

syntax:

```
	switch(condition/expression)
	{
		case val1:
				//code to be execute
				break stmt;
		case val2:
				//code to be execute
				break stmt;
		-
		-
		default:
				//code to be execute if all cases are false.
	}
```

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the option :");
		int opt=sc.nextInt();

		switch(opt)
		{
```

```
                        case 100: System.out.println("It is police number ");
                                        break;
                        case 103: System.out.println("It is enquiry number");
                                        break;
                        case 108: System.out.println("It is emergency number");
                                        break;
                        default:  System.out.println("Invalid option");
                }


        }
}
```

Declaration of break statement in switch case is optional.
If we won't defined break statement then from where our condition is satisfied from there all cases will be executed that state is called fall through state of switch case.

ex:

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter the option :");
                int opt=sc.nextInt();

                switch(opt)
                {
                        case 100: System.out.println("It is police number ");
                                        //break;
                        case 103: System.out.println("It is enquiry number");
                                        //break;
                        case 108: System.out.println("It is emergency number");
                                        //break;
                        default:  System.out.println("Invalid option");
                }


        }
}
```

The allowed datatype of switch case are byte,short,int ,char and String.
If we take other datatypes then we will get compile time error.

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the String :");
		String str=sc.next();

		switch(str)
		{
			case "one": System.out.println("January"); break;
			case "two": System.out.println("February"); break;
			case "three": System.out.println("March"); break;
			case "four": System.out.println("April"); break;
			case "five": System.out.println("May"); break;
			default:  System.out.println("Coming Soon...");
		}


	}
}
```

Q)Write a java program to check given alphabet is a vowel or consonent?

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the Alphabet :");
		char ch=sc.next().charAt(0);
		switch(ch)
		{
			case 'a': System.out.println("It is a vowel"); break;
			case 'e': System.out.println("It is a vowel"); break;
			case 'i': System.out.println("It is a vowel"); break;
			case 'o': System.out.println("It is a vowel"); break;
			case 'u': System.out.println("It is a vowel"); break;
			default:  System.out.println("It is a consonent");
		}

	}
}
```

Assignment
Q)Write a java program to accept six marks of a student then find out total, average and grade?
i) if average is greater then equals to 75 then A grade.
ii)if average is greater then equals to 50 then B grade.
iii)if average is greater then equals to 35 then C grade.
iv) if average is less then 35 then failed.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int m1=89,m2=37,m3=49,m4=53,m5=66,m6=72;

		int total=m1+m2+m3+m4+m5+m6;

		float avg=(float)total/6;

		System.out.println("Total :"+total);
		System.out.println("Average : "+avg);

		if(avg>=75)
			System.out.println("Grade : A grade");
		else if(avg>=50)
			System.out.println("Grade : B grade");
		else if(avg>=35)
			System.out.println("Grade : C grade");
		else
			System.out.println("Grade : Failed");
	}
}
```

3)Iteration statement
Iteration statement is used to execute the code repeatedly.
Iteration statement is possible by using loops.
We have four types of loops.
i) do while loop
ii) while loop
iii) for loop
iv) for each loop

i) do while loop
It will execute the source code untill our condition is true.
syntax:

```
	do
	{
```

```
		-
		- //code to be execute
		-
	}while(condition);
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		do
		{
			System.out.print(i+" "); // infinite 1
		}
		while(i<=10);
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=11;
		do
		{
			System.out.print(i+" ");
		}
		while(i<=10);

	}
}
```

Note:
In do while loop our code will execute atleast for one time either our condition is true or false.

Q)Write a java program to display 10 natural numbers?

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		do
		{
			System.out.print(i+" ");
			i++;
```

```
		}
		while (i<=10);

	}
}
```

Q)Write a java program to display 10 natural numbers in descending order?

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		do
		{
			System.out.print(i+" ");//10 9 8 7 6 5 4 3 2 1
			i--;
		}
		while (i>=1);

	}
}
```

Q)Write a java program to perform sum of 10 natural numbers?

```
class Test
{
	public static void main(String[] args)
	{
		int i=1,sum=0;
		do
		{
			sum=sum+i;
			i++;
		}
		while (i<=10);

		System.out.println("sum of 10 natural numbers is ="+sum);
	}
}
```

Q)Write a java program to find out factorial of a given number?

input: n=5
output: 120 (5*4*3*2*1)

```
import java.util.Scanner;
class Test
{
```

```
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		int i=n,fact=1;
		do
		{
			fact=fact*i;
			i--;
		}
		while (i>=1);

		System.out.println("Factorial of a given number is ="+fact);
	}
}
```

Q)Write a java program to display multiplication table of a given number?

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		int i=1;
		do
		{
			System.out.println(n+" * "+i+" = "+n*i);
			i++;
		}
		while (i<=10);
	}
}
```

ii) while loop

It will execute the source code untill our condition is true.

syntax:

```
while(condition)
	{
```

```
		-
		- //code to be execute
		-
	}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;

		while(i<=10)
		{
			System.out.print(i+" "); //infinite 1
		}
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=11;
		while(i<=10)
		{
			System.out.print(i+" "); //nothing
		}
	}
}
```

Q)Write a java program to display 100 natural numbers?

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		while(i<=100)
		{
			System.out.print(i+" ");
			i++;
		}
	}
}
```

Q)Write a java program to perform sum of 10 natural numbers?

```
class Test
{
	public static void main(String[] args)
	{
		int i=1,sum=0;

		while(i<=10)
		{
			sum=sum+i;
			i++;
		}

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

Q)Write a java program to find out factorial of a given number ?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		int i=n,fact=1;
		while(i>=1)
		{
			fact=fact*i;
			i--;
		}
		System.out.println(fact);
	}
}
```

Q)Write a java program to display multiplication table of a given number?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
```

```
		System.out.println("Enter the number :");
		int n=sc.nextInt(); // 5

		int i=1;
		while(i<=10)
		{
			System.out.println(n+" * "+i+" = "+n*i);
			i++;
		}
	}
}
```

Q)Write a java program to perform sum of digits of a given number?
input: 123
output: 6 (1+2+3)

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt(); // 123

		int rem,sum=0;

		while(n>0)
		{
			rem=n%10;
			sum=sum+rem;
			n=n/10;
		}

		System.out.println("sum of digits of a given number is ="+sum);
	}
}
```

Q)Write a java program to find out given number is armstrong or not?
input: 153 (1*1*1+5*5*5+3*3*3) = (1+125+27) = (153)
output: It is a amrstrong number

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

```
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt(); // 123

		int temp=n;

		int rem,sum=0;

		while(n>0)
		{
			rem=n%10;
			sum=sum+rem*rem*rem;
			n=n/10;
		}

		if(temp==sum)
			System.out.println("It is armstrong number");
		else
			System.out.println("It is not armstrong number");

	}
}
```

Q)Write a java program to find out reverse of a given number?
input: 123
output: 321

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt(); // 123

		int rem,rev=0;
		while(n>0)
		{
			rem=n%10;
			rev=rev*10+rem;
			n=n/10;
		}
```

```
                System.out.println("Reverse of a given number is ="+rev);
        }
}
```

Q)Write a java program to check given number is palindrome or not?
input: 121
output: It is a palindrome number

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter the number :");
                int n=sc.nextInt(); // 123

                int temp=n;

                int rem,rev=0;
                while(n>0)
                {
                        rem=n%10;
                        rev=rev*10+rem;
                        n=n/10;
                }

                if(temp==rev)
                        System.out.println("It is a palindrome number");
                else
                        System.out.println("It is not a palindrome number");
        }
}
```

iii)for loop
It will execute the source code untill our condition is true.
syntax:

```
        for(initialization;condition;incrementation/decrementation)
        {
                -
                - //code to be execute
                -
        }
```

ex:

```
class Test
```

```
{
	public static void main(String[] args)
	{
		for(int i=1;i<=10;i++)
		{
			System.out.print(i+" "); // 1 2 3 4 5 6 7 8 9 10
		}
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=10;i++)
		{
			System.out.print(i+" "); // infinite 1

			i--;
		}
	}
}
```

Q)Write a java program to display sum of even numbers from 1 to 10?

sum = (30) = 2 + 4 + 6 + 8 + 10

```
class Test
{
	public static void main(String[] args)
	{
		int sum=0;
		for(int i=1;i<=10;i++)
		{
			if(i%2==0)
			{
				sum+=i;
			}
		}
		System.out.println("sum of even numbers is ="+sum);
	}
}
```

Q)Write a java program to display number of even and odd numbers from 1 to 10?

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

```
	{
		int even=0,odd=0;
		for(int i=1;i<=10;i++)
		{
			if(i%2==0)
			{
				even++;
			}
			else
			{
				odd++;
			}
		}
		System.out.println("No of evens :"+even);
		System.out.println("No of odds  :"+odd);
	}
}
```

Q)Write a java program to display fibonacci series of a given number?
fibonacci series : 0 1 1 2 3 5 8

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();//5

		int a=0,b=1,c;

		System.out.print(a+" "+b+" ");

		for(int i=2;i<=n;i++)
		{
				c=a+b;
				System.out.print(c+" ");
				a=b;
				b=c;
		}
	}
}
```

Q)Write a java program to check given number is prime or not?
prime numbers:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97……

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();//5

		boolean flag=true;

		for(int i=2;i<=n/2;i++)
		{
			if(n%i==0)
			{
				flag=false;
				break;
			}
		}
		if(flag==true)
			System.out.println("It is a prime number");
		else
			System.out.println("It is not a prime number");
	}
}
```

Q)Write a java program to check given number is perfect or not?
input: 6
output: It is perfect number

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();//5

		int sum=0;
```

```
		for(int i=1;i<n;i++)
		{
			if(n%i==0)
			{
				sum=sum+i;
			}
		}
		if(n==sum)
			System.out.println("It is a perfect number");
		else
			System.out.println("It is not a perfect number");
	}
}
```

Q)Write a java program to find out GCD(Greatest Common Divisor) of two numbers?
input: 12 18
output: 6

```
class Test
{
	public static void main(String[] args)
	{
		int a=12,b=18,gcd=0;

		for(int i=1;i<12 || i<18;i++)
		{
			if(a%i==0 && b%i==0)
			{
				gcd=i;
			}
		}

		System.out.println("GCD of two numbers is ="+gcd);
	}
}
```

Q)Write a java program to display list of prime numbers from 1 to 100?
output: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=2;i<=100;i++)
		{
			boolean flag=true;

			for(int j=2;j<i;j++)
```

```
                {
                        if(i%j==0)
                        {
                                flag=false;
                                break;
                        }
                }
                if(flag==true)
                        System.out.print(i+" ");
        }
    }
}
```

Various ways to declare methods in java

There are four ways to declare methods in java.

1) No returntype With no argument method
2) No returntype With argument method
3) With returntype With no argument method
4) With returntype With argument method

1) No returntype With no argument method

If no arguments then we need to ask input values inside callie method.

Q)Write a java program to perform sum of two numbers using no returntype with no argument method?

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                //caller method
                sum();
        }

        //callie method
        public static void sum()
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter the first number :");
                int a=sc.nextInt();

                System.out.println("Enter the second number :");
                int b=sc.nextInt();

                int c=a+b;
```

```
                System.out.println("sum of two numbers is ="+c);

        }
}
```

3) With returntype With no argument method
A returntype is completely depends upon output datatype.

Q)Write a java program to perform sum of two numbers using with returntype with no argument method?

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                //caller
                int k=sum();
                System.out.println("sum of two numbers is ="+k);
        }
        //callie method
        public static int sum()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the First Number :");
                int a=sc.nextInt();
                System.out.println("Enter the Second Number :");
                int b=sc.nextInt();

                int c=a+b;

                return c;
        }
}
```

2) No returntype With argument method
Arguments are completly depends upon number of inputs.

Q)Write a java program to perform sum of two numbers using no returntype with arguement method?

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
```

```
		System.out.println("Enter the first number :");
		int a=sc.nextInt();

		System.out.println("Enter the second number :");
		int b=sc.nextInt();

		//caller method
		sum(a,b);
	}
	public static void sum(int a,int b)
	{
		int c=a+b;
		System.out.println("sum of two numbers is ="+c);
	}
}
```

4) With returntype With argument method

Q)Write a java program to find out sum of two numbers using with returntype with argument method?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the first number :");
		int a=sc.nextInt();

		System.out.println("Enter the second number :");
		int b=sc.nextInt();

		//caller method
		System.out.println("sum of two numbers is ="+sum(a,b));
	}
	//callie method
	public static int sum(int a,int b)
	{
		int c=a+b;

		return c;
	}
}
```

Assignment

Q)Write a java program to find out factorial of a given number ?

Recursion In Java

A method which called itself for many number of times is called recursion.

Recursion is similar to loopings.

Whenever we use recursion, we should not use loops.

Q) Write a java program to perform sum of two numbers without using arithmetic operator ?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the first number :");
		int a=sc.nextInt();

		System.out.println("Enter the second number :");
		int b=sc.nextInt();

		//caller method
		System.out.println("sum of two numbers is ="+sum(a,b));
	}
	//callie method
	public static int sum(int a,int b)
	{
		if(a==0)
			return b;

		return sum(--a,++b);
	}
}
```

Q)Write a java program to display 10 natural numbers without using loops?

```
class Test
{
	public static void main(String[] args)
	{
		//caller
		display(1);
	}
	//callie method
	public static void display(int i)
	{
		if(i<=10)
		{
			System.out.print(i+" ");
			display(i+1);
```

```
		}
	}
}
```

Q)Write a java program to display Nth element of fibonacci series ?
fibonacci series : 0 1 1 2 3 5 8
input: 4
output: 2

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the number : ");
		int n=sc.nextInt();//4

		//caller method
		System.out.println("Nth element of fibonacci series is ="+fib(n));
	}
	//callie method
	public static int fib(int n)
	{
		if(n==0 || n==1)
			return 0;
		if(n==2)
			return 1;

		return fib(n-1)+fib(n-2);
	}
}
```

Q)Write a java program to find out factorial of a given number using recursion?
input: 5
output: 120

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the number : ");
		int n=sc.nextInt();//5

		//caller
		System.out.println(factorial(n));
```

```
	}
	//callie method
	public static int factorial(int n)
	{
		if(n<0)
			return -1;
		if(n==0)
			return 1;
		return n*factorial(n-1);
	}
}
```

Q)Write a java program to check given number is palindrome or not using recursion?

```
class Test
{
	public static void main(String[] args)
	{
		int num=121;
		int original=num;
		int reversed=0;

		//caller method
		if(isPalindrome(num,original,reversed))
			System.out.println("It is a palindrome number");
		else
			System.out.println("It is not a palindrome number");
	}
	//callie method
	public static boolean isPalindrome(int num,int original,int reversed)
	{
		if(num==0)
			return original==reversed;

		reversed=reversed*10+num%10;

		return isPalindrome(num/10,original,reversed);
	}
}
```

LOOP Patterns

1)

```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

2)

```
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

```
* * * *
* * * *
* * * *
* * * *
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

4)

```
4 4 4 4
3 3 3 3
2 2 2 2
1 1 1 1
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=4;i>=1;i--)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
```

```
	}
}
```

5)

```
A A A A
B B B B
C C C C
D D D D
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(char i='A';i<='D';i++)
		{
			//cols
			for(char j='A';j<='D';j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

6)

```
D D D D
C C C C
B B B B
A A A A
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(char i='D';i>='A';i--)
		{
			//cols
			for(char j='A';j<='D';j++)
			{
				System.out.print(i+" ");
```

```
                }
                //new line
                System.out.println("");
            }
        }
}
```

7)

```
* * * *
*     *
*     *
* * * *
```

```
class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //cols
            for(int j=1;j<=4;j++)
            {
                if(i==1 || i==4 || j==1 || j==4)
                    System.out.print("* ");
                else
                    System.out.print("  ");
            }
            //new line
            System.out.println("");
        }
    }
}
```

8)

```
* - - -
- * - -
- - * -
- - - *
```

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

```
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				if(i==j)
					System.out.print("* ");
				else
					System.out.print("- ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

9)

```
* - - - *
- * - * -
- - * - -
- * - * -
* - - - *
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=5;i++)
		{
			//cols
			for(int j=1;j<=5;j++)
			{
				if(i==j || i+j==6)
					System.out.print("* ");
				else
					System.out.print("- ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

10)

```
  *
  *
* * * * *
  *
  *
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=5;i++)
		{
			//cols
			for(int j=1;j<=5;j++)
			{
				if(i==3 || j==3)
					System.out.print("* ");
				else
					System.out.print("  ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

11)

```
1 1 1
1 0 1
1 1 1
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=3;i++)
		{
			//cols
			for(int j=1;j<=3;j++)
			{
				if(i==2 && j==2)
					System.out.print("0 ");
				else
```

```
                                        System.out.print("1 ");
                        }
                        //new line
                        System.out.println("");
                }
        }
}
```

Left Side Loop patterns

1)

```
1
2 2
3 3 3
4 4 4 4
```

```
class Test
{
        public static void main(String[] args)
        {
                //rows
                for(int i=1;i<=4;i++)
                {
                        //cols
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print(i+" ");
                        }
                        //new line
                        System.out.println("");
                }

        }
}
```

2)

```
1
1 2
1 2 3
1 2 3 4
```

```
class Test
{
        public static void main(String[] args)
        {
                //rows
                for(int i=1;i<=4;i++)
```

```
		{
			//cols
			for(int j=1;j<=i;j++)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

3)

```
*
* *
* * *
* * * *
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

4)

```
4 4 4 4
3 3 3
2 2
1
```

```
class Test
{
```

```
	public static void main(String[] args)
	{
		//rows
		for(int i=4;i>=1;i--)
		{
			//cols
			for(int j=1;j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

5)

```
*
* *
* * *
* * * *
* * *
* *
*
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}
		//descending
		//rows
		for(int i=3;i>=1;i--)
		{
			//cols
```

```
                for(int j=1;j<=i;j++)
                {
                        System.out.print("* ");
                }
                //new line
                System.out.println("");
        }

    }
}
```

6)

```
1
2 3
4 5 6
7 8 9 0
class Test
{
    public static void main(String[] args)
    {
        int k=1;
        //rows
        for(int i=1;i<=4;i++)
        {
            //cols
            for(int j=1;j<=i;j++)
            {
                if(k<=9)
                    System.out.print(k++ +" ");
                else
                    System.out.println("0 ");
            }
            //new line
            System.out.println("");
        }


    }
}
```

Assignment

Q)Write a java program to display below loop pattern?

```
2
3   5
7   11  13
17  19  23 29
```

Right Side Elements

1)

```
   1
 2 2
3 3 3
4 4 4 4
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}

			//elements
			for(int j=1;j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

2)

```
4 4 4 4
 3 3 3
  2 2
   1
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=4;i>=1;i--)
		{
			//space
			for(int j=4;j>i;j--)
```

```
                {
                        System.out.print("  ");
                }

                //elements
                for(int j=1;j<=i;j++)
                {
                        System.out.print(i+" ");
                }
                //new line
                System.out.println("");
        }
    }
}
```

3)

```
    *
   * *
  * * *
 * * * *
  * * *
   * *
    *
```

```
class Test
{
    public static void main(String[] args)
    {
        //ascending
        //rows
        for(int i=1;i<=4;i++)
        {
            //space
            for(int j=4;j>i;j--)
            {
                System.out.print("  ");
            }

            //elements
            for(int j=1;j<=i;j++)
            {
                System.out.print("* ");
            }
            //new line
            System.out.println("");
        }
```

```
		//descending
		//rows
		for(int i=3;i>=1;i--)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}

			//elements
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

Pyramids
========
1)
```
      1
   1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
			//left side element
			for(int j=1;j<=i;j++)
			{
				System.out.print(j+" ");
```

```
                        }
                        //right side elements
                        for(int j=i-1;j>=1;j--)
                        {
                                System.out.print(j+" ");
                        }

                        //new line
                        System.out.println("");
                }
        }
}
```

2)

```
1 2 3 4 3 2 1
  1 2 3 2 1
    1 2 1
      1
```

```
class Test
{
        public static void main(String[] args)
        {
                //rows
                for(int i=4;i>=1;i--)
                {
                        //space
                        for(int j=4;j>i;j--)
                        {
                                System.out.print("  ");
                        }
                        //left side element
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print(j+" ");
                        }
                        //right side elements
                        for(int j=i-1;j>=1;j--)
                        {
                                System.out.print(j+" ");
                        }

                        //new line
                        System.out.println("");
                }
        }
}
```

3)

```
    *
   * * *
  * * * * *
* * * * * * *
  * * * * *
   * * *
    *
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
			//left side element
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//right side elements
			for(int j=i-1;j>=1;j--)
			{
				System.out.print("* ");
			}

			//new line
			System.out.println("");
		}

		//descending
		//rows
		for(int i=3;i>=1;i--)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
```

```
                        //left side element
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print("* ");
                        }
                        //right side elements
                        for(int j=i-1;j>=1;j--)
                        {
                                System.out.print("* ");
                        }

                        //new line
                        System.out.println("");
                }
        }
}
```

Interview Question

Q)Write a java program to display below loop pattern?

```
1            1
1 2        2 1
1 2 3    3 2 1
1 2 3 4 4 3 2 1
```

```
class Test
{
        public static void main(String[] args)
        {
                int rows=4;

                //rows
                for(int i=1;i<=rows;i++)
                {
                        //left side elements
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print(j+" ");
                        }
                        //space
                        for(int j=1;j<=(rows-i)*2;j++)
                        {
                                System.out.print("  ");
                        }
                        //right side elements
                        for(int j=i;j>=1;j--)
                        {
                                System.out.print(j+" ");
```

```
                }
                //new line
                System.out.println("");
            }

        }
}
```

Q)Write a java program to display below loop pattern?

```
1
2 1
1 2 3
4 3 2 1
class Test
{
        public static void main(String[] args)
        {
                int rows=4;

                for(int i=1;i<=rows;i++)
                {
                        if(i%2==0)
                        {
                                for(int j=i;j>=1;j--)
                                {
                                        System.out.print(j+" ");
                                }
                        }
                        else
                        {
                                for(int j=1;j<=i;j++)
                                {
                                        System.out.print(j+" ");
                                }
                        }
                        //new line
                        System.out.println("");
                }

        }
}
```

4)Jump Statement
Jump statement is used to jump from one section of code to another section.
We have two jump statements in java.

i) break statement
ii) continue statement

i) break statement
It is used to break the execution of loops and switch case.
For conditional statement we can use if condition.

syntax: break;

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		break;
		System.out.println("stmt2");
	}
}
```

o/p:

break outside switch or loop

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(true)
		{
			break;
		}
		System.out.println("stmt2");
	}
}
```

o/p:

break outside switch or loop

ex:3

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=10;i++)
		{
			if(i==5)
```

```
                        {
                                break;
                        }
                        System.out.print(i+" ");//1 2 3 4
                }
        }
}
```

ii) continue statement
It is used to continue the execution of loops.
For conditional statements we can use if condition.

syntax: continue;

ex:1

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("stmt1");
                continue;
                System.out.println("stmt2");
        }
}
```

o/p: continue outside of loop

ex:2

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("stmt1");
                if(true)
                {
                        continue;
                }
                System.out.println("stmt2");
        }
}
```

o/p: continue outside of loop

ex:3

```
class Test
{
```

```
	public static void main(String[] args)
	{
		for(int i=1;i<=10;i++)
		{
			if(i==5)
			{
				continue;
			}
			System.out.print(i+" ");//1 2 3 4 6 7 8 9 10
		}
	}
}
```

Interview Program

Q)Write a java program to display reverse of a given number in words?

input: 123

output: ThreeTwoOne

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt();

		while(n>0)
		{
			switch(n%10)
			{
				case 0 : System.out.print("Zero");break;
				case 1 : System.out.print("One");break;
				case 2 : System.out.print("Two");break;
				case 3 : System.out.print("Three");break;
				case 4 : System.out.print("Four");break;
				case 5 : System.out.print("Five");break;
				case 6 : System.out.print("Six");break;
				case 7 : System.out.print("Seven");break;
				case 8 : System.out.print("Eight");break;
				case 9 : System.out.print("Nine");break;
			}
			n=n/10;
		}
	}
}
```

Assignment
Write a java program to display sum of digits of a given number and display in reverse order?
input: 987
output: 42

Arrays
A normal variable we can store only one value at a time.
In order to store more then one value in a single variable then we need to use arrays.
Array is a collection of homogeneous data elements.
The main advantages of arrays are
1) We can represent multiple elements using single variable name.
ex: int[] arr={10,20,30};
2) Performance point of view arrays are recommanded to use.
The main disadvantages of arrays are.
1) Arrays are fixed in size once if we create an array there is no chance of increasing or decreasing the size of an array.
2) To use array concept in advanced we should know what is the size of an array which is always not possible.
In java, arrays are divided into three types.
1)Single dimensional array
2)Double dimensional array / two dimensional array
3)Multi dimensional array / three dimensional array

Array declaration
At the time of array declaration we should not specify array size.

Arrays

Single dimensional array | Double dimensional array | Multidimensional array

Single dimensional array	Double dimensional array	Multidimensional array
int[] arr;	int[][] arr;	int[][][] arr;
int []arr;	int [][]arr;	int [][][]arr;
int arr[];	int arr[][];	int arr[][][];
	int[] []arr;	int[][] []arr;
	int[] arr[];	int[][] arr[];
	int []arr[];	int[] [][]arr;
		int[] arr[][];
		int[] []arr[];
		int []][]arr[];
		int []arr[][];

Array Creation
In java, every array consider as an object.Hence we will use new operator to create an array.
ex: int[] arr=new int[3];

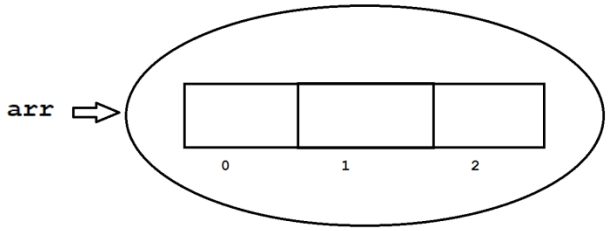


Diagram : java20.1

Rules to construct an array

Rule1:

At the time of array creation compulsary we need to specify array size.

ex: int[] arr=new int[3]; //valid

int[] arr=new int[]; //C.T.E array dimension missing

Rule2:

It is legal to have an array size with zero.

ex: int[] arr=new int[0];

System.out.println(arr.length);//0

Rule3:

We can't give negative number as an array size otherwise we will get NegativeArraySizeException.

ex: int[] arr=new int[-3]; // R.E NegativeArraySizeException

Rule4:

The allowed datatype for an array size is byte,short,int and char .If we take other datatypes then we will get compile time error.

ex: byte b=10;

int[] arr=new int[b];

int[] arr=new int['a'];

int[] arr=new int[10.5f]; //invalid

Rule5:

The maxmimum length we can take for an array is maximum length of int.

ex: int[] arr=new int[2147483647];

Array Initialization

Once if we create an array then every array element will be initialized with default values.

It is possible to change default values with customized values also.

ex:

```
int[] arr=new int[3];
arr[0]=10;
arr[1]=20;
arr[2]=30;
arr[3]=40; // R.E ArrayIndexOutOfBoundsException
```

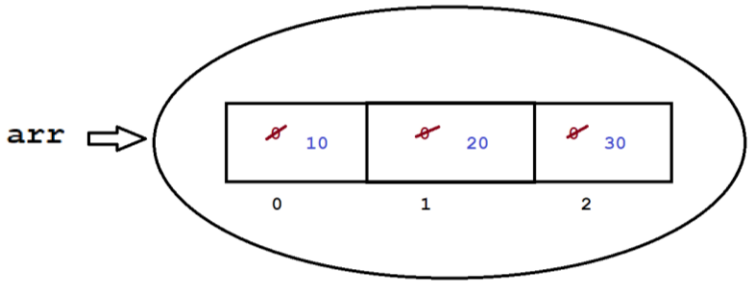


Diagram: java20.2

Array Declaration , Creation and Initialization using single line

```
int[] arr;
arr = new int[3];
arr[0]=10;
arr[1]=20;
arr[2]=30;      ====>       int[] arr={10,20,30};
                ======>     char[] carr={'a','b','c'};
                ======>     String[] sarr={"hi","hello","bye"};
```

Q)What is the difference between length and length() ?

length

It is a final variable which is applicable for arrays.

It will return size of an array.

```
ex:     class Test
        {
                public static void main(String[] args)
                {
                        int[] arr=new int[3];
                        System.out.println(arr.length);//3
                }
        }
```

length()

It is a predefined method applicable for String objects.

It will return number of characters present in String.

```
ex:     class Test
        {
                public static void main(String[] args)
                {
                        String str="bhaskar";

                        System.out.println(str.length());//7
                }
        }
```

Q)Write a java program to accept array elements and display them ?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the array size :");
		int size=sc.nextInt();//4

		int[] arr=new int[size];

		//inserting elements
		for(int i=0;i<arr.length;i++)
		{
			System.out.println("Enter the element :");
			arr[i]=sc.nextInt();
		}

		//display elements
		for(int i=0;i<arr.length;i++)
		{
			System.out.print(arr[i]+" ");
		}

	}
}
```

Q)Write a java program to display array elements ?

input: arr = 4 8 1 3 9

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,3,9};

		//normal for loop
		for(int i=0;i<arr.length;i++)
		{
			System.out.print(arr[i]+" ");
		}
		System.out.println("\n==========");

		//for each loop
		for(int ele:arr)
```

```
		{
			System.out.print(ele+" ");
		}
	}
}
```

Q)Write a java program to perform sum array elements ?
input: arr = 4 8 1 3 9
output: 25

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

		int[] arr={4,8,1,3,9};

		//sum of array elements
		int sum=0;
		for(int ele:arr)
		{
			sum+=ele;
		}

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

Q)Write a java program to display array elements in reverse order?
input: arr = 4 8 1 3 9
output: 9 3 1 8 4

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,3,9};

		for(int i=arr.length-1;i>=0;i--)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

Q)Write a java program to display array elements in ascending/sorting order?
input: arr = 4 8 1 3 9
output: 1 3 4 8 9

approach1:

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{

		int[] arr={4,8,1,3,9};

		Arrays.sort(arr); // 1 3 4 8 9

		//display elements
		for(int ele:arr)
		{
			System.out.print(ele+" ");
		}
	}
}
```

Approach2

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

		int[] arr={4,8,1,3,9};

		//ascending logic
		for(int i=0;i<arr.length;i++)
		{
			for(int j=0;j<arr.length;j++)
			{
				if(arr[i]<arr[j])
				{
					int temp=arr[i];
					arr[i]=arr[j];
					arr[j]=temp;
				}
			}
		}

		//display elements
		for(int ele:arr)
		{
```

```
                System.out.print(ele+" ");
            }
        }
}
```

Q)Write a java program to display array elements in descending order?
input: arr = 4 8 1 3 9
output: 9 8 4 3 1
approach1:

```
import java.util.Arrays;
class Test
{
        public static void main(String[] args)
        {

            int[] arr={4,8,1,3,9};

            Arrays.sort(arr);//1 3 4 8 9

            //display elements
            for(int i=arr.length-1;i>=0;i--)
            {
                System.out.print(arr[i]+" ");
            }
        }
}
```

approach2

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

            int[] arr={4,8,1,3,9};

            //descending logic
            for(int i=0;i<arr.length;i++)
            {
                for(int j=0;j<arr.length;j++)
                {
                    if(arr[i]>arr[j])
                    {
                        int temp=arr[i];
```

```
                                        arr[i]=arr[j];
                                        arr[j]=temp;
                                }
                        }
                }

                //display elements
                for(int ele:arr)
                {
                        System.out.print(ele+" ");
                }
        }
}
```

Q)Write a java program to display least element from given array?
input: arr = 4 8 1 3 9
output: 1
approach1

```
import java.util.Arrays;
class Test
{
        public static void main(String[] args)
        {
                int[] arr={4,8,1,3,9};

                Arrays.sort(arr);//1 3 4 8 9

                System.out.println(arr[0]);
        }
}
```

Approach2

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

                int[] arr={4,8,1,3,9};

                int small=arr[0];

                for(int i=0;i<arr.length;i++)
                {
                        if(arr[i]<small)
```

```
                {
                        small=arr[i];
                }
        }
        System.out.println(small);
    }
}
```

Q)Write a java program to display highest element from given array?
input: arr = 4 8 1 3 9
output: 9
approach1

```
import java.util.Arrays;
class Test
{
    public static void main(String[] args)
    {
        int[] arr={4,8,1,3,9};

        Arrays.sort(arr);

        System.out.println(arr[arr.length-1]);//9
    }
}
```

approach2

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

        int[] arr={4,8,1,3,9};

        int big=arr[0];

        for(int i=0;i<arr.length;i++)
        {
            if(arr[i]>big)
            {
                big=arr[i];
            }
        }
        System.out.println(big);
    }
}
```

Q)Write a java program to display duplicate elements from given array?
input: 2 4 5 7 2 3 9 3 1 1
output: 2 3 1

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={2,4,5,7,2,3,9,3,1,1};

		//duplicate elements
		for(int i=0;i<arr.length;i++)
		{
			for(int j=i+1;j<arr.length;j++)
			{
				if(arr[i]==arr[j])
					System.out.print(arr[i]+" ");
			}
		}

	}
}
```

Q)Write a java program to find out unique elements from given array?
input: 2 3 5 6 7 2 9 3 1 5
output: 6 7 9 1

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={2,3,5,6,7,2,9,3,1,5};
		//unique elements
		for(int i=0;i<arr.length;i++)
		{
			int cnt=0;
			for(int j=0;j<arr.length;j++)
			{
				if(arr[i]==arr[j])
				{
					cnt++;
				}
			}
			if(cnt==1)
				System.out.print(arr[i]+" ");
		}
	}
}
```

Q)Write a java program to find out most repeating element from given array?
input: 1 3 5 2 1 9 1 1 3 3 7 1 6
output: 1 is repeating for 5 times

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={1,3,5,2,1,9,1,1,3,3,7,1,6};

		int maxCount=0;
		int element=0;

		for(int i=0;i<arr.length;i++)
		{
			int cnt=0;
			for(int j=0;j<arr.length;j++)
			{
				if(arr[i]==arr[j])
				{
					cnt++;
				}
			}
			if(maxCount<cnt)
			{
				maxCount=cnt;
				element=arr[i];
			}
		}
		System.out.println(element+" is repeating for "+maxCount+" times");
	}
}
```

Q)Write a java program to display prime elements from given arrays?
input: 5 9 2 6 21 25 29
output: 5 2 29

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,9,2,6,21,25,29};

		for(int ele:arr)
		{
			boolean flag=true;

			for(int j=2;j<=ele/2;j++)
```

```
                        {
                                if(ele%j==0)
                                {
                                        flag=false;
                                        break;
                                }
                        }
                        if(flag==true)
                                System.out.print(ele+" ");
                }
        }
}
```

Q)Write a java program to segregate 0's and 1's?
input: 1 0 1 1 0 0 1 0 0 1
output: 0 0 0 0 0 1 1 1 1 1
approach1

```
import java.util.Arrays;
class Test
{
        public static void main(String[] args)
        {
                int[] arr={1,0,1,1,0,0,1,0,0,1};

                Arrays.sort(arr);

                for(int ele:arr)
                {
                        System.out.print(ele+" ");
                }
        }
}
```

approach2

```
class Test
{
        public static void main(String[] args)
        {
                int[] arr={1,0,1,1,0,0,1,0,0,1};

                int[] resArr=new int[arr.length];

                //inserting 0's
                int j=0;
```

```
		for(int ele:arr)
		{
			if(ele==0)
			{
				resArr[j++]=0;
			}
		}

		//inserting 1's
		while(j<arr.length)
		{
			resArr[j++]=1;
		}

		//display elements
		for(int ele:resArr)
		{
			System.out.print(ele+" ");
		}
	}
}
```

Q)Write a java program to find out missing element from given array?
input: 5 1 3 2 6 7
output: 4

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,1,3,2,6,7};

		int sum_of_arr_ele=arr.length+1;

		int sum=(sum_of_arr_ele*(sum_of_arr_ele+1))/2;

		for(int ele:arr)
		{
			sum=sum-ele;
		}

		System.out.println("Missing element is ="+sum);
	}
}
```

Q)Write a java program to find out leader element from given array?
input: 2 10 6 34 19 1 7
output: 7 19 34

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={2,10,6,34,19,1,7};

		int max=arr[arr.length-1];

		System.out.print(max+" ");

		for(int i=arr.length-2;i>=0;i--)
		{
			if(arr[i]>max)
			{
				max=arr[i];
				System.out.print(max+" ");
			}
		}
	}
}
```

Q)Write a java program to perform sum of array elements?
input: arr1 = 2 4 7 1 3
arr2 = 9 8 6 4 3
output: 11 12 13 5 6

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr1 ={2,4,7,1,3};
		int[] arr2 ={9,8,6,4,3};
		int[] resArr=new int[arr1.length];
		for(int i=0;i<arr1.length;i++)
		{
			resArr[i]=arr1[i]+arr2[i];
		}
		//for each loop
		for(int ele:resArr)
		{
			System.out.print(ele+" ");
		}
	}
}
```

Q)Write a java program to delete first occurance of a given element?

input: arr = 2 4 5 6 4 9 1 3 4
ele = 4

output: 2 5 6 4 9 1 3 4

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={2,4,5,6,4,9,1,3,4};
		int ele=4;

		int[] resArr=new int[arr.length-1];

		int cnt=0;
		int j=0;
		for(int i=0;i<arr.length;i++)
		{
			if(arr[i]==ele && cnt==0)
			{
				cnt++;
				continue;
			}
			resArr[j++]=arr[i];
		}

		//display
		for(int i:resArr)
		{
			System.out.print(i+" ");
		}

	}
}
```

Q)write a java program to concatinate two arrays and display them in sorting order?

input: arr1 = 9 6 8 10 7
arr2 = 1 5 4 2 3

output: 1 2 3 4 5 6 7 8 9 10

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr1 ={9,6,8,10,7};
		int[] arr2 ={1,5,4,2,3};

		int size1=arr1.length;
```

```
		int size2=arr2.length;

		arr1=Arrays.copyOf(arr1,size1+size2);

		int j=0;
		for(int i=size1;i<arr1.length;i++)
		{
			arr1[i]=arr2[j++];
		}

		Arrays.sort(arr1);

		//display the elements
		for(int ele:arr1)
		{
			System.out.print(ele+" ");
		}
	}
}
```

Q)Write a java program to insert given element on given index?

input: arr = 5 8 1 4 6 9
ele = 100
index = 3

output: 5 8 1 100 4 6 9

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr ={5,8,1,4,6,9};
		int ele = 100;
		int index = 3;

		arr=Arrays.copyOf(arr,arr.length+1);

		for(int i=arr.length-1;i>=index;i--)
		{
			arr[i]=arr[i-1];
		}

		arr[index]=ele;

		//display
		for(int i:arr)
		{
```

```
                System.out.print(i+" ");
            }
        }
}
```

Two Dimensional Array

Two dimensional array is a combination of rows and columns.
Two dimensional array is implemented based on array of arrays approach but not in matrix form.
The main objective of two dimensional array is memory utilization.
Two dimensional array is used to develop business oriented applications, gaming applications and matrix type of applications.
We can declare two dimensional array as follow.

syntax: datatype[][] variable_name= new int[rows][cols];
ex: int[][] arr=new [3][3];
Here we can store 9 elements.

Q)Write a java program to display array elements in matrix form?

```
import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the rows :");
        int rows=sc.nextInt();//3

        System.out.println("Enter the cols :");
        int cols=sc.nextInt();//3

        int[][] arr=new int[rows][cols];

        //insert elements
        for(int i=0;i<rows;i++)
        {
            for(int j=0;j<cols;j++)
            {
                System.out.println("Enter the element :");
                arr[i][j]=sc.nextInt();
            }
        }

        //display elements
        for(int i=0;i<rows;i++)
        {
            for(int j=0;j<cols;j++)
```

```
                {
                        System.out.print(arr[i][j]+" ");
                }
                //new line
                System.out.println("");
        }

    }
}
```

Q)Write a java program to perform sum of diagonal elements?

```
class Test
{
    public static void main(String[] args)
    {
        int[][] arr={{1,2,3}, {4,5,6}, {7,8,9}};
        //display elements
        int sum=0;
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                if(i==j)
                {
                    sum=sum+arr[i][j];
                }
            }
        }
        System.out.println("sum of diagonal elements is ="+sum);

    }
}
```

Q)Write a java program to display sum of upper triangle elements?

```
class Test
{
    public static void main(String[] args)
    {
        int[][] arr={{1,2,3}, {4,5,6}, {7,8,9}};
        //display elements
        int sum=0;
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                if(i<j)
```

```
                        {
                                sum=sum+arr[i][j];
                        }
                }
        }
        System.out.println("sum of upper triangle elements is ="+sum);

    }
}
```

Q)Write a java program to display sum of lower triangle elements?

```
class Test
{
    public static void main(String[] args)
    {
        int[][] arr={{1,2,3}, {4,5,6}, {7,8,9}};
        //display elements
        int sum=0;
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                if(i>j)
                {
                    sum=sum+arr[i][j];
                }
            }
        }
        System.out.println("sum of lower triangle elements is ="+sum);
    }
}
```

Q)Write a java program to display array elements in spiral form ?

```
input:  1 2 3
        4 5 6
        7 8 9
output: 1 2 3 6 9 8 7 4 5
public class Test
{
   public static void main(String[] args)
   {
      int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
      int rows = matrix.length;
      int cols = matrix[0].length;
```

```
int top = 0;
int bottom = rows - 1;
int left = 0;
int right = cols - 1;

while (true)
            {
   if (left > right)
                    {
      break;
   }

   // Print top row
   for (int i = left; i <= right; i++) {
      System.out.print(matrix[top][i] + " ");
   }
   top++;

   if (top > bottom) {
      break;
   }

   // Print right column
   for (int i = top; i <= bottom; i++) {
      System.out.print(matrix[i][right] + " ");
   }
   right--;

   if (left > right) {
      break;
   }

   // Print bottom row
   for (int i = right; i >= left; i--)
                    {
      System.out.print(matrix[bottom][i] + " ");
   }
   bottom--;

   if (top > bottom) {
      break;
   }

   // Print left column
   for (int i = bottom; i >= top; i--)
      {
```

```
            System.out.print(matrix[i][left] + " ");
        }
        left++;
        }//while loop
    }
}
```

Anonymous Array

Sometimes we will declare an array without name such type of nameless arrays are called anonymous array.

The main objective of anonymous array is just for instance use.

We can declare anonymous array as follow.

ex: new int[]{10,20,30};
new int[][]{{10,20,30},{40,50,60}};

```
public class Test
{
   public static void main(String[] args)
   {
                //caller method
                sum(new int[]{10,20,30});

        }
        //callie method
        public static void sum(int[] arr)
        {
                int sum=0;
                for(int i:arr)
                {
                        sum+=i;
                }
                System.out.println(sum);
        }
}
```

ex:2

```
public class Test
{
   public static void main(String[] args)
   {
                //caller method
                System.out.println(sum(new int[]{10,20,30}));

        }
        //callie method
        public static int sum(int[] arr)
```

```
	{
		int sum=0;
		for(int i:arr)
		{
			sum+=i;
		}
		return sum;
	}
}
```

OOPS - OOPS stands for Object Oriented Programming System/Structure.

Object oriented technology

A technology which provides very good environment to represent our data in the form of objects is called object oriented technology.

A technology is said to be object oriented if it support following features.

ex: class, object, Abstraction, Encapsulation, Inheritance & Polymorphism

class

A class is a collection of data members and behaviours in a single unit.

Here data members means variables, fields and properties.

Here behaviours means methods, actions and characteristics.

In general , a class is a collection of variables and methods.

It is a blue print of an object.

We can declare a class as follow.

syntax:

```
	optional
	   |
	modifier class class_name <extends> Parent_class <implements> Interface_name
	{
		-
		- // variables & methods
		-
	}
```

A class will accept following modifiers.

ex: default, public, final & abstract

Q)What is the difference between default class and public class?

default class

If we declare any class as default then we can access that class within the package.

```
ex:	class A
	{
		-
		-
	}
```

public class

If we declare any class as public then we can access that class within the package and outside of the package.

ex:
```
public class A
{
        -
        -
}
```

Q)What is **final class**?

If we declare any class as final creating child class is not possible.

or

If we declare any class as final then extending someother class is not possible.

ex:
```
final class A
{

}
class B extends A  // invalid
{
}
```

Q)What is **abstract class**?

If we declare any class as abstract then creating object for that class is not possible.

ex:
```
abstract class A
{
        -
        -
}
A a=new A(); //invalid
```

object

It is a instance of a class.

Allocating memory for our data members is called instance.

Object is a outcome of a blue print.

Memory space will be created when we create an object.

We can declare object as follow.

syntax: class_name reference_variable=new constructor();

ex: `Test t = new Test();`

It is possible to declare more then one object in a single class.

```
class Test
{
        public static void main(String[] args)
        {
                Test t1=new Test();
                Test t2=new Test();
                Test t3=new Test();
```

```
                System.out.println(t1.hashCode());
                System.out.println(t2.hashCode());
                System.out.println(t3.hashCode());

                System.out.println(t1); //Test@Hexavalue
                System.out.println(t2.toString());
                System.out.println(t3.toString());
        }
}
```

hashCode()

A hashCode() method present in Object class.
For every object , JVM will create a unique identification number i.e hash code.
In order to read hash code of an object we need to use hashCode() method.

```
Test t=new Test();
```

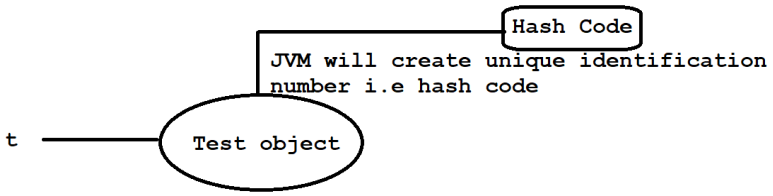


Diagram: java23.1

toString()

A toString() method present in Object class.
Whenever we are trying to display any object reference directly or indirectly toString() method will be executed.

Object class

Object is a class which is present in java.lang package.
It is a parent class for every java class.
Object class contains following methods.
ex: cmd> javap java.lang.Object

hashCode()
notify()
notifyAll()
getClass()
toString()
equals()
and etc.

Data Hiding

Our internal data should not go out directly.
It means outside person must not access our data directly.
Using private mmodifier we can achieve data hiding concept.
The main objective of data hiding is to provide security.

ex: class Account
{
private double balance;
-
-
}

Abstraction

Hiding internal implementation and highlighting the set of services is called abstraction.
Using abstract classes and interfaces we can implement Abstraction.
Best example of abstraction is GUI(Graphical User Interface) ATM machine where bank people will hide internal implementation and highlights the set of services like banking ,withdrawl, mini statement and etc.
The main advantages of abstraction are.
1)It gives security because it will hide internal implementation from the outsider.
2) Enhancement becomes more easy because without effecting enduser they can perform any changes in our internal system.
3) It provides flexibility to the enduser to use the system.
4) It improve maintainability of an application

Encapsulation

The process of encapsulating or grouping variables and it's associate methods in a single entity is called encapsulation.

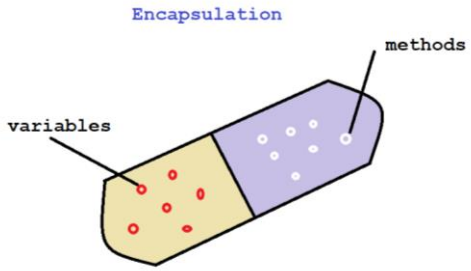


Diagram: java23.2

A class is said to be encapsulated class if it supports data hiding and abstraction.
In encapsulation,for every variable we need to write setter and getter method.

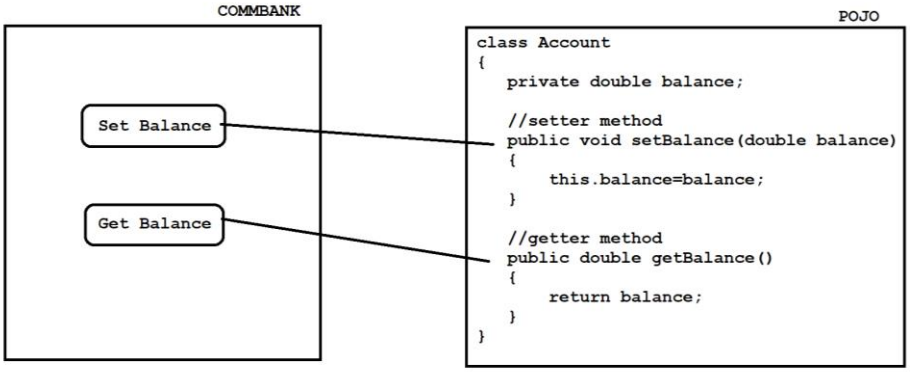


Diagram: java23.3

The main advantages of encapsulation are
1)It gives security.
2)Enhancement becomes more easy.
3)It provides flexibility to the enduser.
4)It improves maintainability of an application.
The main disadvantage of encapsulation is , it will increase the length of our code and slowdown the execution process.

Approach1

```
class Student
{
	private int studId;
	private String studName;
	private double studFee;

	//setter methods
	public void setStudId(int studId)
	{
		this.studId=studId;
	}
	public void setStudName(String studName)
	{
		this.studName=studName;
	}
	public void setStudFee(double studFee)
	{
		this.studFee=studFee;
	}

	//getter methods
	public int getStudId()
	{
		return studId;
	}
	public String getStudName()
	{
		return studName;
	}
	public double getStudFee()
	{
		return studFee;
	}

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

```
                s.setStudId(101);
                s.setStudName("Alan");
                s.setStudFee(1000.0d);
                System.out.println("Student Id :"+s.getStudId());
                System.out.println("Student Name :"+s.getStudName());
                System.out.println("Student Fee :"+s.getStudFee());

        }
}

Approach2
class Student
{
        private int studId;
        private String studName;
        private double studFee;

        //setter methods
        public void setStudId(int studId)
        {
                this.studId=studId;
        }
        public void setStudName(String studName)
        {
                this.studName=studName;
        }
        public void setStudFee(double studFee)
        {
                this.studFee=studFee;
        }

        //getter methods
        public int getStudId()
        {
                return studId;
        }
        public String getStudName()
        {
                return studName;
        }
        public double getStudFee()
        {
                return studFee;
        }
}
class Test
```

```
{
	public static void main(String[] args)
	{
		Student s=new Student();
		s.setStudId(101);
		s.setStudName("Alan");
		s.setStudFee(1000.0d);
		System.out.println("Student Id :"+s.getStudId());
		System.out.println("Student Name :"+s.getStudName());
		System.out.println("Student Fee :"+s.getStudFee());

	}
}
```

*Note
Abstraction is used to hide the data.
Encapsulation is used to protected the data using access modifiers.

Interview Questions

Q)What is tightly encapsulated class?

A class is said to be tightly encapsulated class.If all the variables of that class must be private and we don't need to check these variables have setter or getter methods or not.

ex:
```
class A
{
	private int i=10;
}
```

ex:
```
class A
{
	private int i=10;
}
class B extends A
{
	private int j=20;
}
```

Q)What is the difference between POJO class and Java Bean class?

POJO class

POJO stands for Plain Old Java Object.
A class is said to be pojo class if it supports following two properties.
1)All variables must be private.
2)All the variables must have setter and getter methods.

JavaBean class

A class is said to be java bean class if it supports following four properties.
1) A class should be public.
2) A class should have zero-argument constructor.
3) All variables must be private.

4) All variables must have setter and getter methods.
*Note: Every java bean class is a pojo class but every pojo class is not a java bean class.

Is-A relationship

Is-A relationship is also known as inheritance.
By using extends keyword we can implements Is-A relationship.
The main objective of Is-A relationship is to provide reusability.

```
class Parent
{
	public void m1()
	{
		System.out.println("Parent-M1 Method");
	}
}
class Child extends Parent
{
	public void m2()
	{
		System.out.println("Child-M2 Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.m1();
		Child c=new Child();
		c.m1();
		c.m2();
		Parent p1=new Child();
		p1.m1();
		//Child c1=new Parent(); //invalid
	}
}
```

Inheritance

Inheritance is a mechanism where we will derive a class in the presence of existing class.

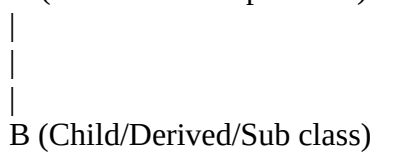
or

Inheritance is a mechanism where one class will inherit the properties of another class.
We have five types of inheritance.
1)Single level inheritance
2)Multi level inheritance
3)Multiple inheritance
4)Hierarchical inheritance
5)Hybrid inheritance

1)Single level inheritance

If we derived a class in the presence of one base class is called single level inheritance.

Diagram:




```
class A
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();
		B b=new B();
		b.m1();
		b.m2();
	}
}
```

2)Multi level inheritance

If we derived a class in the presence of one base class and that class will derived from another base class is called multi level inheritance.

Diagram:


```
class A
{
```

```
	public void m1()
	{
		System.out.println("M1-Method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
class C extends B
{
	public void m3()
	{
		System.out.println("M3-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();
		B b=new B();
		b.m1();
		b.m2();
		C c=new C();
		c.m1();
		c.m2();
		c.m3();
	}
}
```

3)Multiple inheritance

In java a class can't extends more then one class simultenously because java does not support multiple inheritance.

```
ex:	class A
	{}
	class B
	{}
	class C extends A,B  --> Invalid
	{
	}
```

But interface can extends more then one interface simultenously so we can achieve multiple inheritance concept through interfaces.

```
ex:     interface A
        {}
        interface B
        {}
        interface C extends A,B  --> valid
        {}
```

If our class does not extends any other class then our class is a direct child class of Object class.

```
ex:                         Diag:
        class A                     Object
        {                           |
        }                           |
                                    A
```

If our class extends some other class then our class is a indirect child class of Object class.

```
ex:                         Diag:
        class A                     Object
        {                           |
        }                           |
        class B extends A           A
        {                           |
        }                           |
                                    B
```

Java does not support cyclic inheritance.

```
ex:     class A extends B
        {
        }
        class B extends A
        {
        }
```

Q)Why java does not support multiple inheritance?

There may chance of raising ambiguity problem that's why java does not support multiple inheritance.

```
ex:     p1.m1()                         p2.m1()
          |_____________________________________|
                          |
                        c.m1()
```

4)Hierarchical inheritance

If we derived multiple classes in the presence of one base class is called hierarchical inheritance.

```
Diagram:            A
                    |
                |---------------|
                B               C
```

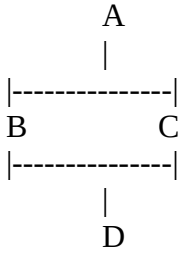
```
class A
{
```

```
	public void m1()
	{
		System.out.println("M1-Method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
class C extends A
{
	public void m3()
	{
		System.out.println("M3-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();

		B b=new B();
		b.m1();
		b.m2();

		C c=new C();
		c.m1();
		c.m3();
	}
}
```

5)Hybrid inheritance

Hybrid inheritance is a combination of more then one inheritance.
Java does not support hybrid inheritance.

Diagram:



Has-A relationship

Has-A relationship is also known as Composition and Aggregation.

There is no specific keyword to implements Has-A relationship but mostly we will use new operator.

The main objective of Has-A relationship is to provide reusability.

Has-A relationship will increase dependency between two components.

```
ex:	class Engine
	{
		-
		- //engine specific functionality
		-
	}
	class Car
	{
		Engine e=new Engine();
	}

class Ihub
{
	public String courseName()
	{
		return "Full Stack Java With AWS";
	}
	public double courseFee()
	{
		return 30000d;
	}
	public String trainerName()
	{
		return "Niyaz Sir";
	}
}
class Usha
{
	public void getCourseDetails()
	{
		Ihub i=new Ihub();
		System.out.println("Course Name :"+i.courseName());
		System.out.println("Course Fee :"+i.courseFee());
		System.out.println("Trainer Name :"+i.trainerName());
	}
}
class Student
{
	public static void main(String[] args)
```

```
        {
                Usha u=new Usha();
                u.getCourseDetails();
        }
}
```

Composition

Without existing container object there is no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

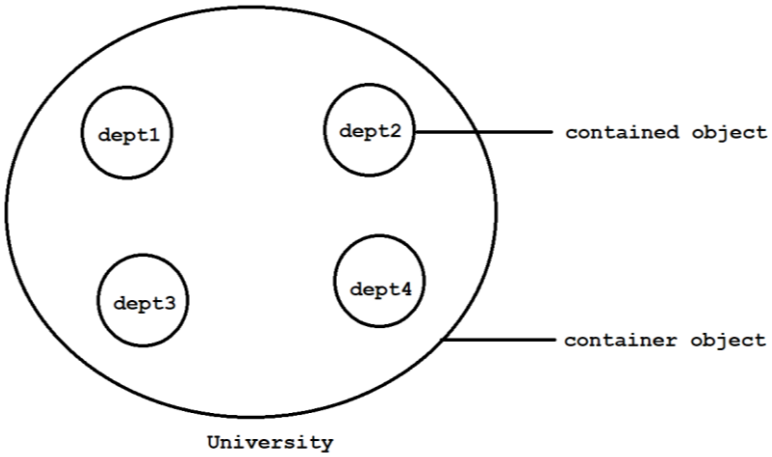


Diagram: java24.1

Aggregation

Without existing container object there is a chance of having contained object then the relationship between container and contained object is called aggregation which is loosely association.

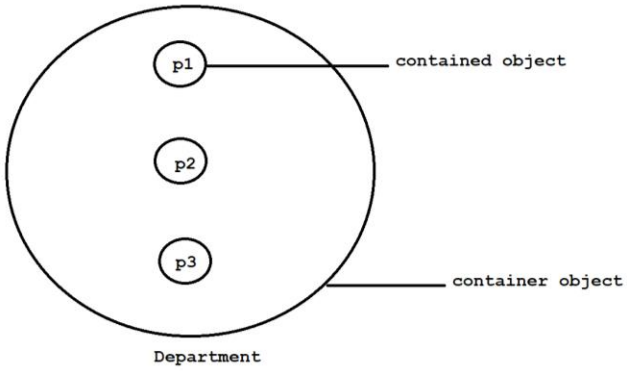


Diagram: java24.2

Method overloading

Having same method name with different parameters in a single class is class method overloading.

All the methods present in a class are called overloaded methods.

Method overloading will reduce complexity of the programming.

```
class MeeSeva
{
	//overloaded methods
	public void search(int voterId)
	{
		System.out.println("Details found using voterId");
	}
	public void search(String houseNo)
	{
		System.out.println("Details found using house No");
	}
	public void search(long aadharNo)
	{
		System.out.println("Details found using aadhar No ");
	}
}
class Test
{
	public static void main(String[] args)
	{
		MeeSeva ms=new MeeSeva();
		ms.search(101);
		ms.search("1-6-4/1/A");
		ms.search(20002l);
	}
}
```

Method overriding

Having same method name with same parameters in a two different classes is called method overriding.

Methods which are present in parent class are called overridden methods.

Methods which are present in child class are called overriding methods.

```
class Parent
{
	//overridden methods
	public void property()
	{
		System.out.println("Cash+Gold+Land");
	}
	public void marry()
	{
		System.out.println("Subhalakshmi");
	}
}
class Child extends Parent
{
```

```
	//overriding methods
	public void marry()
	{
		System.out.println("Rashmika");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.property(); // Cash+Gold+Land
		p.marry(); //subhalakshmi

		Child c=new Child();
		c.property(); // Cash+Gold+Land
		c.marry(); // Rashmika

		Parent p1=new Child();
		p1.property(); // Cash+Gold+Land
		p1.marry(); // Rashmika

	}
}
```

If we declare any method as final then overriding is not possible.

ex:

```
class Parent
{
	//overridden methods
	public void property()
	{
		System.out.println("Cash+Gold+Land");
	}
	public final void marry()
	{
		System.out.println("Subhalakshmi");
	}
}
class Child extends Parent
{
	//overriding methods
	public void marry()
	{
		System.out.println("Rashmika");
	}
}
```

```
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.property(); // Cash+Gold+Land
		p.marry(); //subhalakshmi

		Child c=new Child();
		c.property(); // Cash+Gold+Land
		c.marry(); // Rashmika

		Parent p1=new Child();
		p1.property(); // Cash+Gold+Land
		p1.marry(); // Rashmika

	}
}
```

Method Hiding

Method hiding is exactly same as method overriding with following differences.

Method overriding	Method Hiding
All the methods present in method overriding must be non-static.	All the methods present in method Hiding must be static.
Method resolution will taken care by a JVM based on runtime object.	Method resolution will taken care by compiler based on reference type.
It is also known as Runtime polymorphism, Dynamic polymorphism, late binding.	It is also known as compile time polymorphism, Static polymorphism, early binding.

```
class Parent
{
	//overridden methods
	public static void property()
	{
		System.out.println("Cash+Gold+Land");
	}
	public static void marry()
	{
		System.out.println("Subhalakshmi");
	}
}
class Child extends Parent
{
```

```
	//overriding methods
	public static void marry()
	{
		System.out.println("Rashmika");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.property(); // Cash+Gold+Land
		p.marry(); //subhalakshmi

		Child c=new Child();
		c.property(); // Cash+Gold+Land
		c.marry(); // Rashmika

		Parent p1=new Child();
		p1.property(); // Cash+Gold+Land
		p1.marry(); // Subhalakshmi

	}
}
```

Interview Questions

Q) Can we overload main method in java?

Yes, we can overload main method in java but JVM always execute main method with String[ ] parameter only.

```
class Test
{
	public static void main(int[] iargs)
	{
		System.out.println("int argument");
	}
	public static void main(String[] args)
	{
		System.out.println("String argument");
	}
}
```

Q)Can we override main method in java?

No , We can't override main method in java because it is static.

Polymorphism

Polymorphism has taken from Greek word.
Here poly means many and morphism means forms.
The ability to represent in different forms is called polymorphism.
The main objective of polymorphism is to provide flexbility.

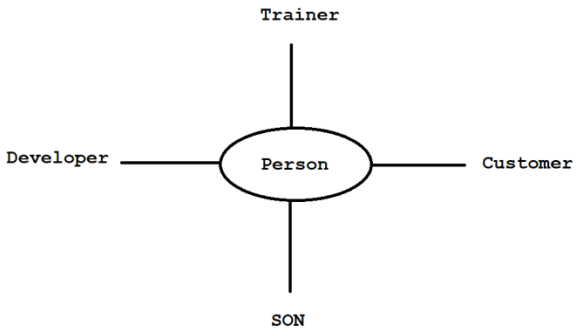


Diagram: java25.1

In java polymorphism is divided into two types.
1)Compile time polymorphism / static polymorphism / early binding
2)Runtime polymorphism / dynamic polymorphism / late binding

1)Compile time polymorphism

A polymorphism which exhibits at compile time is called compile time polymorhpism.
ex: Method Overloading
Method Hiding

2)Runtime polymorphism

A polymorphism which exhibits at runtime is called runtime polymorphism.
ex: Method overriding

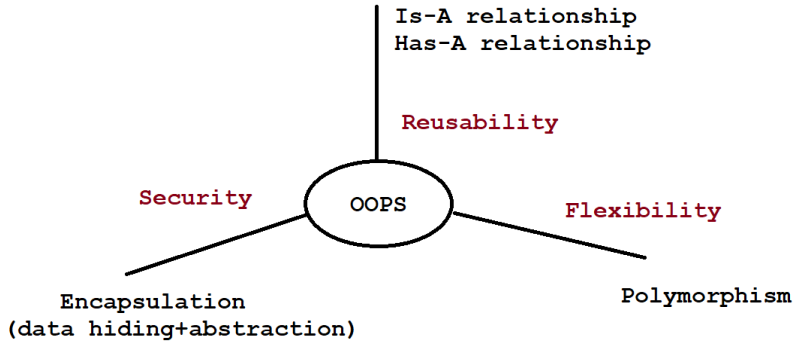


Diagram: java25.2

Constructors

A constructor is a special method which is used to initialized an object.
Having same name as class name is called constructor.
A constructor does not allow any returntype.
A constructor will execute when we create an object.
A constructor will accept following modifiers.
ex: default, public, private & protected
In java constructors are divided into two types.
1)Userdefined constructor
2)Default constructor

1)Userdefined constructor

A constructor which is created by the user based on the application requirement is called userdefined constructor.

It is classified into two types.

i) Zero-Argument constructor

ii) Parameterized constructor

i) Zero-Argument constructor

Suppose if we won't pass any argument to userdefined constructor then that constructor is called zero-argument constructor.

```
class Test
{
	Test()
	{
		System.out.println("0-arg const");
	}
	public static void main(String[] args)
	{
		System.out.println("Main-Method");
	}
}
```

o/p: Main-Method

```
class Test
{
	Test()
	{
		System.out.println("0-arg const");
	}
	public static void main(String[] args)
	{
		System.out.println("Main-Method");
		Test t=new Test();
	}
}
```

o/p: Main-Method
0-arg const

```
class Test
{
	public Test()
	{
		System.out.println("0-arg const");
	}
	public static void main(String[] args)
	{
```

```
		Test t1=new Test();
		System.out.println("Main-Method");
		Test t2=new Test();
	}
}
```
o/p: 0-arg const
Main-Method
0-arg const

ii) Parameterized constructor

Suppose if we are passing atleast one argument to userdefined constructor then that constructor is called parameterized constructor.

```
class Employee
{
	private int empId;
	private String empName;
	private double empSal;

	//parameterized constructor
	public Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}
	public void getEmployeeDetails()
	{
		System.out.println("Employee Id :"+empId);
		System.out.println("Employee Name :"+empName);
		System.out.println("Employee Salary :"+empSal);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Employee e=new Employee(101,"Alan Morries",1000d);
		e.getEmployeeDetails();
	}
}
```

2)Default constructor

It is a compiler generated constructor for every java program where we are not defining atleast zero argument constructor.

Default constructor is a empty implementation.

We can see default constructor by using below command.

ex: javap -c Test

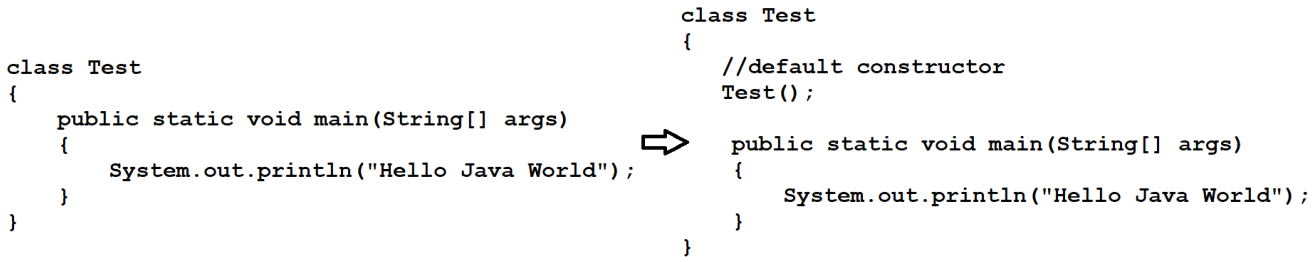


Diagram: java25.3

this keyword

A this keyword is a java keyword which is used to refer current class object reference.

We can utilize this keyword in following ways.

i) To refer current class variables

ii)To refer current class methods

iii)To refer current class constructors

i) To refer current class variables

```
class A
{
        int i=10;
        int j=20;
        A(int i,int j)
        {
                System.out.println(i+" "+j); // 100 200
                System.out.println(this.i+" "+this.j); //10  20
        }
}
class Test
{
        public static void main(String[] args)
        {
                A a=new A(100,200);
        }
}
```

ii) To refer current class methods

```
class A
{
        public void m1()
        {
                System.out.println("M1 method");
                this.m2();
        }
        public void m2()
        {
                System.out.println("M2 method");
```

```
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();
	}
}
```

iii) To refer current class constructor

```
class A
{
	A()
	{
		System.out.println("0-arg const");
	}
	A(int i)
	{
		this();
		System.out.println("int arg const");
	}
	A(double d)
	{
		this(10);
		System.out.println("double arg const");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A(10.5d);
	}
}
```

super keyword

A super keyword is a java keyword which is used to refer super class object reference.
We can utilize super keyword in following ways.
i) To refer super class variables
ii) To refer super class methods
iii) To refer super class constructors

i) To refer super class variables

```
class A
```

```
{
	int i=10;
	int j=20;
}
class B extends A
{
	int i=100;
	int j=200;
	B(int i,int j)
	{
		System.out.println(this.i+" "+this.j); // 100 200
		System.out.println(super.i+" "+super.j); //10 20
		System.out.println(i+" "+j); // 1000 2000
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B(1000,2000);
	}
}
```

ii) To refer super class methods

```
class A
{
	public void m1()
	{
		System.out.println("M1 method");
	}
}
class B extends A
{
	public void m2()
	{
		super.m1();
		System.out.println("M2 method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B();
		b.m2();
	}
```

```
}
```

iii) To refer super class constructors

```
class A
{
        A()
        {
                System.out.println("A const");
        }
}
class B extends A
{
        B()
        {
                super();
                System.out.println("B const");
        }
}
class Test
{
        public static void main(String[] args)
        {
                new B();
        }
}
```

Interfaces

An interface is a collection of zero or more abstract methods.
Abstract methods are incomplete methods because they ends with semicolon and do not have any body.
It is not possible to create object for interfaces.
To write the implementation of abstract methods of an interface we will use implementation class.
It is possible to create object for implementation class.
By default every abstract method is a public and abstract.
Interface contains only constants i.e public static final.

syntax:

```
interface <interface_name>
        {
                -
                - //abstract methods
                - //constants
                -
        }
```

If we know Service Requirement Specification then we need to use interface.

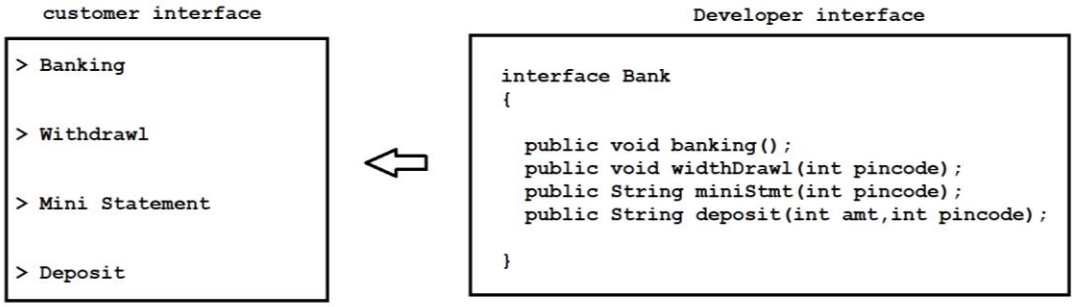


Diagram: java26.1

```
interface A
{
	//abstract method
	public abstract void m1();
}
class B implements A
{
	public void m1()
	{
		System.out.println("M1 method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
	}
}
```

ex:

```
interface A
{
	//abstract method
	public abstract void m1();
}

class Test
{
	public static void main(String[] args)
	{
		//Anonymous inner class
		A a=new A()
		{
			public void m1()
```

```
                {
                        System.out.println("M1 method");
                }
        };
        a.m1();
    }
}
```

If interface contains four methods then we need to override all methods otherwise we will get compile time error.

```
interface A
{
    //abstract methods
    public abstract void show();
    public void display();
    abstract void view();
    void see();
}
class B implements A
{
    public void show()
    {
        System.out.println("show-method");
    }
    public void display()
    {
        System.out.println("display-method");
    }
    public void view()
    {
        System.out.println("view-method");
    }
    public void see()
    {
        System.out.println("see-method");
    }

}

class Test
{
    public static void main(String[] args)
    {
        A a=new B();
        a.show();
        a.display();
```

```
                a.see();
                a.view();
        }
}
```

In java, a class can't extends more then one class but interface can extends more then one interface.

```
interface A
{
        void m1();
}
interface B
{
        void m2();
}
interface C extends A,B
{
        void m3();
}

class D implements C
{
        public void m1()
        {
                System.out.println("M1-method");
        }
        public void m2()
        {
                System.out.println("M2-method");
        }
        public void m3()
        {
                System.out.println("M3-method");
        }

}
class Test
{
                        C c =new D();
                        c.m1();
        public static void main(String[] args)
        {
                        c.m2();
                        c.m3();
        }
}
```

A class can implements more then one interface.

```
interface Father
{
	//constant
	float HT=6.2f;
	void height();
}
interface Mother
{
	float HT=5.8f;
	void height();
}
class Child implements Father,Mother
{
	public void height()
	{
		float height=(Father.HT+Mother.HT)/2;
		System.out.println("Child Height :"+height);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Child c=new Child();
		c.height();
	}
}
```

Java 8

According java 8 version, interface is a collection of default methods, static methods and abstract methods.

Marker interface

Interface which does not have any constants or abstract methods is called marker interface.
Emptry interface is called marker interface.
Using marker interface we will get some ability to do.
We have following list of marker interface.

ex: Serializable
Cloneable
Remote
and etc.

Abstract classes

Abstract class is a collection of zero or more abstract methods and concrete methods.
A abstract keyword is applicable for methods and classes but not for variables.
It is not possible to create object for abstract class.

To write the implementation of abstract methods of a abstract class we will use sub classes.
By default every abstract method is a public and abstract.
Abstract class contains only instance variables.
syntax: abstract class class_name
```
{
        -
        - // instance variables
        - // abstract methods
        - // concrete methods
        -
}
```
If we know partial implementation then we need to use Abstract class.

```
abstract class Plan
{
        //instance variable
        protected double rate;

        //abstract method
        public abstract void getRate();

        //concrete method
        public void calculateBillAmt(int units)
        {
                System.out.println("Total Units :"+units);
                System.out.println("Total Bill :"+ rate*units);
        }
}
class DomesticPlan extends Plan
{
        public void getRate()
        {
                rate=2.5d;
        }
}
class CommercialPlan extends Plan
{
        public void getRate()
        {
                rate=5.0d;
        }
}
class Test
{
        public static void main(String[] args)
        {
                DomesticPlan dp=new DomesticPlan();
```

```
                dp.getRate();
                dp.calculateBillAmt(250);

                CommercialPlan cp=new CommercialPlan();
                cp.getRate();
                cp.calculateBillAmt(250);
        }
}
```

Q)What is the difference between interface and abstract class?

interface	abstract class
To declare interface we will use interface keyword.	To declare abstract class we wil use abstract keyword.
Interface is a collection of abstract methods,default methods and static methods.	It is a collection of abstract methods and concrete methods.
It does not allow constructor.	It allows constructor.
It does not allow blocks.	It allows blocks.
To write the implementation of abstract methods we will use implementation class.	To write the implementation of abstract methods we will use sub classes.
It contains only constants.	It contains only instance variables.
Multiple inheritence is possible.	Multiple inheritance is not possible.
If we know only specification then we need to use interface.	If we know partial implementation then we need to use abstract class.

API

API stands for Application Programming Interface.
It is a base for the programmer to develop software applications.
API is a collection of packages.
We have three types of API's.

1)Predefined API

Built-In API is called predefined API.

ex: https://docs.oracle.com/javase/8/docs/api/

2)Userdefined API

API which is created by the user based on the application requirement.

3)Third party API

API which is given by third party vendor.

ex: JAVAZOOM API
iText API
and etc.

Packages

Package is a collection of classes ,interfaces , enums and Annotations.
Here enum is a special class and Annotation is a special interface.
In general, a package is a collection of classes and interfaces.

Package is a also known as folder or a directory.
In java, we have two types of packages.
1)Predefined packages
2)Userdefined packages

1)Predefined packages

Built-In packages are called predefined packages.

ex: java.lang
java.io
java.util
java.text
java.util.stream
java.sql
javax.servlet
and etc.

2)Userdefined packages

Packages which are created by the user based on the application requirement are called userdefined packages.
We can declare userdefined package as follow.
syntax: package <package_name>;

```
package com.ihub.www;
import java.util.Calendar;
class Test
{
	public static void main(String[] args)
	{
		Calendar c=Calendar.getInstance();

		//convert time to 24 hours
		int h=c.get(Calendar.HOUR_OF_DAY);
		if(h<12)
			System.out.println("Good Morning");
		else if(h<16)
			System.out.println("Good Afternoon");
		else if(h<20)
			System.out.println("Good Evening");
		else
			System.out.println("Good Night");
	}
}
```

We can compile the code by using below command.

```
ex:                    current directory
                              |
        javaprog> javac  -d   .   Test.java
                           |
                    Destination folder
```

We can run the code by using below command.

ex: javaprog> java com.ihub.www.Test

| |

pkg name classname

Singleton class

A class which allows us to create only one object is called singleton class.

Using a class if we call any method and that method returns same class object is called singleton class.

We have following list of singleton class.

ex: Calendar
LocalTime
LocalDate
and etc.

To create our own singleton class we need to use private constructor and factory method.

```
class Singleton
{
	static Singleton singleton=null;

	//private constructor
	private Singleton()
	{

	}
	//factory method
	public static Singleton getInstance()
	{
		if(singleton==null)
		{
			singleton=new Singleton();
		}
		return singleton;
	}
}
class Test
{
	public static void main(String[] args)
	{
		Singleton s1=Singleton.getInstance();
		System.out.println(s1.hashCode());

		Singleton s2=Singleton.getInstance();
		System.out.println(s2.hashCode());
	}
}
```

Inner classes

Sometimes we will declare a class inside another class such concept is called inner class.

```
ex:     class  <Outer_Class>
        {
                class <Inner_Class>
                {
                        -
                        - //code to be declare
                        -
                }
        }
```

Inner classes introduced as a part of event handling to remove GUI bugs.

Because of powerful features and benefits of inner classes, programmer started to use inner class in our regular programming.

In inner class , we can't declare static members.

Accessing inner class data from static area of outer class

```
class Outer
{
        class Inner
        {
                public void m1()
                {
                        System.out.println("M1-Method");
                }
        }
        public static void main(String[] args)
        {
                Outer.Inner i=new Outer().new Inner();
                i.m1();
        }
}
```

Note:

If we compile above program we will get two .class files i.e

Outer.class and Outer$Inner.class

ex:

```
class Outer
{
        class Inner
        {
                public void m1()
                {
                        System.out.println("M1-Method");
                }
        }
```

```
	public static void main(String[] args)
	{
		new Outer().new Inner().m1();
	}
}
```

Accessing inner class data from non-static area of outer class

```
class Outer
{
	class Inner
	{
		public void m1()
		{
			System.out.println("M1-Method");
		}
	}
	public void m2()
	{
		Inner i=new Inner();
		i.m1();
	}
	public static void main(String[] args)
	{
		Outer o=new Outer();
		o.m2();
	}
}
```

Enum

Enum is a group of named constants.
Enum concept introduced in 1.5v.
Using enum we can create our own datatype called enumerated datatype.
When compare to old language enum , java enum is more powerful.

```
syntax: enum type_name
	{
		val1,val2,..valN
	}
ex:	enum  Months
	{
		JAN,FEB,MAR
	}
```

Internal implementation of enum

Every enum internally implements as final class concept and extends with java.lang.Enum class.
Every enum constant is a reference variable of enum type.

```
enum Months          public final class Months extends java.lang.Enum
{                    {
JAN,FEB,MAR  =>      public static final Months JAN=new Months();
}                            public static final Months FEB=new Months();
                             public static final Months MAR=new Months();
                     }
```

Declaration and Usage of enum

```
enum Months
{
        JAN,FEB,MAR
}
class Test
{
        public static void main(String[] args)
        {
                Months m=Months.JAN;
                System.out.println(m);//JAN
        }
}
```

ex:

```
enum Months
{
        JAN,FEB,MAR
}

class Test
{
        public static void main(String[] args)
        {
                Months m=Months.FEB;
                switch(m)
                {
                        case JAN: System.out.println("January"); break;
                        case FEB: System.out.println("February"); break;
                        case MAR: System.out.println("March"); break;
                }
        }
}
```

java.lang.Enum class

Power to enum will be inherited from java.lang.Enum class.
It contains following two methods.

1)values() It is a static method which returns group of constants from enum.
2)ordinal() It returns ordinal number of enum.

```
enum Week
{
	MON,TUE,WED,THU,FRI,SAT,SUN
}

class Test
{
	public static void main(String[] args)
	{
		Week[] w=Week.values();

		//for each loop
		for(Week w1:w)
		{
			System.out.println(w1+" ---------- "+w1.ordinal());
		}
	}
}
```

When compare to old language enum , java enum is more powerful because in addition to constants we can declare variables, constructors and methods.

```
enum Cloths
{
	SILK,KHADI,COTTON;

	Cloths()
	{
		System.out.println("constructor");
	}
}

class Test
{
	public static void main(String[] args)
	{
		Cloths c=Cloths.SILK;
	}
}
```

ex:

```
enum Cloths
{
	SILK,KHADI,COTTON;

	static int i=10;
```

```
	public static void main(String[] args)
	{
		System.out.println(i);//10
	}
}
```

Wrapper classes

The main objective of wrapper classes are

1) To wrap primitive type to wrapper object and vice versa.

2) To defined several utility methods.

Primitive type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

constructor

Every wrapper class contains two constructors.One will take corresponding primitive as an argument and another will take corresponding String as an argument.

Wrapper class	constructor
Byte	byte or String
Short	short or String
Integer	int or String
Long	long or String
Float	float or String
Double	double or String
Boolean	boolean or String
Character	char

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(10);
		System.out.println(i1);//10

		Integer i2=new Integer("20");
		System.out.println(i2);//20
	}
}
```

```
class Test
{
	public static void main(String[] args)
	{
		Boolean b1=new Boolean(true);
		System.out.println(b1);//true

		Boolean b2=new Boolean("false");
		System.out.println(b2);//false
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Character c=new Character('a');
		System.out.println(c); //a
	}
}
```

Utility methods

1) valueOf()

It is similar to constructor.

It is used to convert primitive type to wrapper object.

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=Integer.valueOf(10);
		System.out.println(i1);//10

		Long l1=Long.valueOf("20");
		System.out.println(l1);//20
	}
}
```

2) xxxValue()

It will convert wrapper object to primitive type.

```
class Test
{
	public static void main(String[] args)
	{
		Integer i=new Integer(10);
```

```
		byte b=i.byteValue();
		System.out.println(b);

		short s=i.shortValue();
		System.out.println(s);
	}
}
```

3) parseXxx()
It is used to convert string type to primitive type.

```
class Test
{
	public static void main(String[] args)
	{
		String str="100";

		int i=Integer.parseInt(str);
		System.out.println(i); //100

		long l=Long.parseLong(str);
		System.out.println(l); //100

		float f=Float.parseFloat(str);
		System.out.println(f); //100.0
	}
}
```

4) toString()
It is used to convert wrapper object type to String type.

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(10);

		String s=i1.toString();

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

Q)Write a java program to perform sum of two binary numbers?
input: 1010
0101
output: 1111
ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the first binary number :");
		String binary1=sc.next(); //1010

		System.out.println("Enter the second binary number :");
		String binary2=sc.next(); //0101

		//converting binary to decimal
		int a=Integer.parseInt(binary1,2);
		int b=Integer.parseInt(binary2,2);

		int c=a+b;

		//converting decimal to binary
		String result=Integer.toBinaryString(c);
		System.out.println("sum of two binary numbers is ="+result);
	}
}
```

Q)Types of objects in java?
We have two types of objects in java.
1)Immutable object
2)Mutable object

1)Immutable object
After object creation if we perform any changes then for every change a new object will be created such type of object is called immutable object.
ex: String and Wrapper classes
2)Mutable object
After object creation if we perform any changes then all the required changes will be done in a same object such type of object is called mutable object.
ex: StringBuffer and StringBuilder

String
String is a set of characters which is enclosed in a double quotation.
String is a immutable object.
case1: After object creation we can't perform any changes.If we perform any changes then for every change a new object will be created such behaviour is called immutability of an object.
ex: String s=new String("bhaskar");
s.concat("solution");

System.out.println(s); // bhaskar

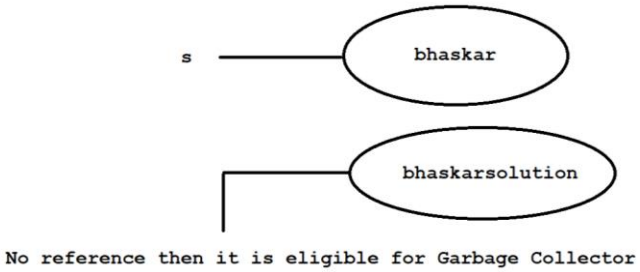


Diagram: java28.1

case2:

Q)What is the difference between == and .equals() method?

==

It is a equality operator or comparision operator which always returns boolean value.

It is used to reference comparision or address comparision.

```
class Test
{
	public static void main(String[] args)
	{
		String s1=new String("bhaskar");
		String s2=new String("bhaskar");

		System.out.println(s1==s2);//false
	}
}
```

.equals()

It is a method present in String class which always returns boolean value.

It is used for content comparision which is case sensitive.

```
class Test
{
	public static void main(String[] args)
	{
		String s1=new String("bhaskar");
		String s2=new String("bhaskar");

		System.out.println(s1.equals(s2));//true
	}
}
```

case3: Once if we create a String object, two objects will be created.One is on heap and another is on SCP (String Constant Pool) area.But 's' always points to heap area only.

```
String s=new String("bhaskar");
```

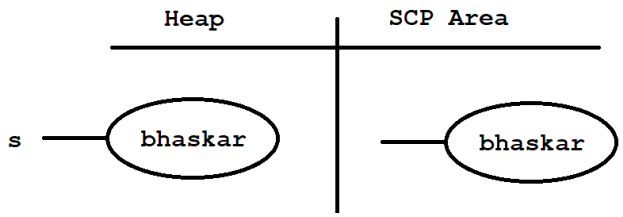


Diagram: java28.2

Object creation in SCP area is always optional.First JVM Will check is there any object is created with same content or not.If it is created then JVM simply refers to that object.If it is not created then JVM will create a new object.HEnce there is no chance of having duplicate objects in SCP area.

Even though SCP object do not have any reference, Garbage collector can't access them.

SCP objects will destroy when JVM shutdowns or terminated.

```
String s1=new String("bhaskar");
String s2=new String("bhaskar");
String s3="bhaskar";
String s4="bhaskar";
String s5="solution";
```

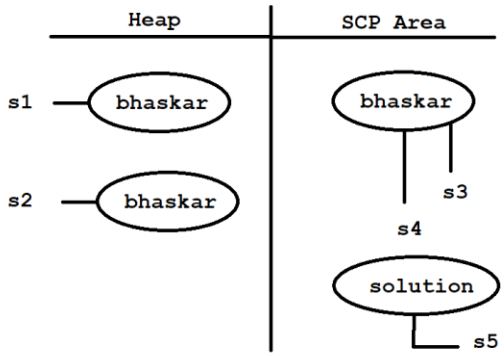


Diagram: java29.1

Interning of String object

With the help of heap object reference if we want corresponding SCP object reference then we need to use intern() method.

```
String s1=new String("bhaskar");

String s2=s1.intern();

String s3="bhaskar";

System.out.println(s2==s3);//true
```

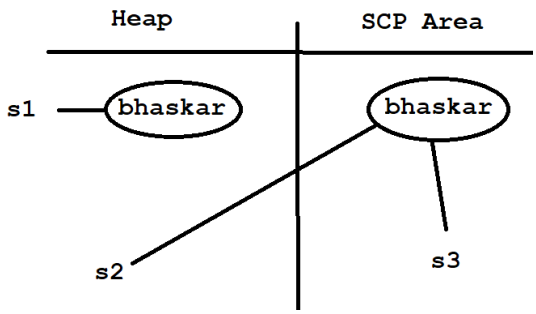


Diagram: java29.2

String important methods

Q)Write a java program to accept one string and dispaly it?

```
import java.util.Scanner;
class Test
{
```

```
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the string :");
		String str=sc.next();

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

approach2

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the string :");
		String str=sc.nextLine();

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

Q)Write a java program to remove all special characters from given string?

input: Ihub@Talent_M$angeme#nt515;

output: IhubTalentManageent515

```
class Test
{
	public static void main(String[] args)
	{
		String str="Ihub@Talent_M$angeme#nt515";

		str=str.replaceAll("[^A-Za-z0-9]","");

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

Q)Write a java program to find out length of the string?

input: hello

output: 5

```
class Test
{
```

```
	public static void main(String[] args)
	{
		String str="hello";

		System.out.println(str.length());//5
	}
}
```

Q)Write a java program to convert uppercase letter to lowercase letter?
input: IHUBTALENT
output: ihubtalent

```
class Test
{
	public static void main(String[] args)
	{
		String str="IHUBTALENT";

		str=str.toLowerCase();

		System.out.println(str);//ihubtalent
	}
}
```

Q)Write a java program to convert lowercase letter to uppercase letter?
input: ihubtalent
output: IHUBTALENT

```
class Test
{
	public static void main(String[] args)
	{
		String str="ihubtalent";

		str=str.toUpperCase();

		System.out.println(str);//IHUBTALENT
	}
}
```

Q)Write a java program to find out given the strings are equal or not?
input: str1 = "ihub";
str2 = "ihub";
output: Both are equals

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

```
                String str1="ihub";
                String str2="ihub";

                if(str1.equals(str2))
                        System.out.println("Both are equals");
                else
                        System.out.println("Both are not equal");
        }
}
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                String str1="ihub";
                String str2="IHUB";

                if(str1.equalsIgnoreCase(str2))
                        System.out.println("Both are equals");
                else
                        System.out.println("Both are not equal");
        }
}
```

Q)Write a java program to check given word present in a string or not?

input: str = "Welcome to ihub talent management"
word = "ihub"

output: It is available

```
class Test
{
        public static void main(String[] args)
        {
                String str="Welcome to ihub talent management";

                String word="ihub";

                if(str.contains(word))
                        System.out.println("It is available");
                else
                        System.out.println("It is not available");
        }
}
```

Q)Write a java program to display the character from string which is present in given index?

input: str = "ihubtalent"

index = 5

output: a

```
class Test
{
	public static void main(String[] args)
	{
		String str="ihubtalent";

		int index =5;

		System.out.println(str.charAt(index));//a
	}
}
```

Q)Write a java program to find out index of first occurance character in a given string?

input: str = "ihubtalentmanagement"
ch='a';

output: 5

```
class Test
{
	public static void main(String[] args)
	{
		String str = "ihubtalentmanagement";

		char ch='a';

		System.out.println(str.indexOf(ch));
	}
}
```

Q)Write a java program to find out index of last occurance character in a given string?

input: str = "ihubtalentmanagement"
ch='a';

output: 13

```
class Test
{
	public static void main(String[] args)
	{
		String str = "ihubtalentmanagement";

		char ch='a';

		System.out.println(str.lastIndexOf(ch));
	}
}
```

Q)Write a java program to find out number of occurance of a given character?

input: str = "ihubtalentmanagement"
char ch='a';

output: 3 times

```
class Test
{
	public static void main(String[] args)
	{
		String str = "ihubtalentmanagement";

		char ch='a';

		int cnt=0;
		for(int i=0;i<str.length();i++)
		{
			char alphabet=str.charAt(i);

			if(alphabet==ch)
			{
				cnt++;
			}
		}

		System.out.println(cnt+" times");
	}
}
```

Q)Write a java program to display the string in reverse order?

input: hello

output: olleh

```
class Test
{
	public static void main(String[] args)
	{
		String str="hello";

		char[] carr=str.toCharArray();

		String rev="";
		for(int i=carr.length-1;i>=0;i--)
		{
			rev+=carr[i];
		}
		System.out.println(rev);//olleh
	}
}
```

Q)Write a java program to check given string is palindrome or not?
input: madam
output: It is palindrome string

```
class Test
{
	public static void main(String[] args)
	{
		String str="madam";

		char[] carr=str.toCharArray();

		String rev="";
		for(int i=carr.length-1;i>=0;i--)
		{
			rev+=carr[i];
		}

		if(str.equals(rev))
			System.out.println("It is palindrome string");
		else
			System.out.println("It is not palindrome string");
	}
}
```

Q)Write a java program to display reverse of a sentence?
input: This Is Java Class
output: Class Java Is This

```
class Test
{
	public static void main(String[] args)
	{
		String str="This Is Java Class";

		String[] sarr=str.split(" ");

		String rev="";
		for(int i=sarr.length-1;i>=0;i--)
		{
			rev+=sarr[i]+" ";
		}

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

Q)Write a java program to display reverse of a word in a sentence?
input: This Is Java Class
output: sihT sI avaJ ssalC

```
class Test
{
	public static void main(String[] args)
	{
		String str="This Is Java Class";

		String[] sarr=str.split(" ");   // This    Is    Java    Class


		//for each loop
		String rev="";
		for(String s:sarr)
		{
			char[] carr=s.toCharArray();

			for(int i=carr.length-1;i>=0;i--)
			{
				rev+=carr[i];
			}
			//add space
			rev+=" ";
		}

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

Q)Write a java program to find out duplicate characters from given string?
input: google
output: go

```
class Test
{
	public static void main(String[] args)
	{
		String str="google";

		String characters="";
		String duplicates="";

		for(int i=0;i<str.length();i++)
		{
			String current=Character.toString(str.charAt(i));
```

```
                if(characters.contains(current))
                {
                        if(!duplicates.contains(current))
                        {
                                duplicates+=current;
                                continue;
                        }
                }
                characters+=current;
        }

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

Q)Write a java program to find out unique/distinct characters from given string?
input: google
output: gole

```
class Test
{
    public static void main(String[] args)
    {
        String str="google";

        String characters="";
        String duplicates="";

        for(int i=0;i<str.length();i++)
        {
            String current=Character.toString(str.charAt(i));

            if(characters.contains(current))
            {
                if(!duplicates.contains(current))
                {
                    duplicates+=current;
                    continue;
                }
            }
            characters+=current;
        }

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

Assignments

Q)Write a java program to number of uppercase letters, lowercase letters, digits , words and spaces?

input: This Is Java Class 29

output: uppercase : 4
lowercase : 11
digits : 2
words : 5
spaces : 4

Q)Write a java program to display the string starting with uppecase letter?

input: This is Java class for Freshers

output: This Java Freshers

Q)Write a java program to display the strings which are palindrome?

input: racar is madam for dad

output: racar madam dad

Q)Write a java program to display the string below format?

input: A1B2C3D4

output: ABBCCCDDDD

```
class Test
{
	public static void main(String[] args)
	{
		String str="A1B2C3D4";

		for(int i=0;i<str.length();i++)
		{
			if(Character.isAlphabetic(str.charAt(i)))
			{
				System.out.print(str.charAt(i));
			}
			else
			{
				int a=Character.getNumericValue(str.charAt(i)); // 1 2 3 4

				for(int j=1;j<a;j++)
				{
					System.out.print(str.charAt(i-1));
				}
			}
		}
	}
}
```

Q)Write a java program to find most repeating character in a given string?
input: googleformat
output: o is repeating for 3 times

```
class Test
{
	public static void main(String[] args)
	{
		String str="googleformat";

		int maxCount=0;

		char element=' ';

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

			for(int j=0;j<str.length();j++)
			{
				if(str.charAt(i)==str.charAt(j))
				{
					cnt++;
				}
			}

			if(maxCount<cnt)
			{
				maxCount=cnt;
				element=str.charAt(i);
			}
		}
		System.out.println(element+" is repeating for "+maxCount+" times");

	}
}
```

Q)Write a java program to perform right rotation of a string?
input: str = "ihubtalent";
cnt = 2;
output: ubtalentih

```
class Test
{
	public static void main(String[] args)
	{
		String str="ihubtalent";
		int cnt=2;
```

```
                char[] carr=str.toCharArray();          // i h u b t a l e n t

                String str1=str.substring(cnt,carr.length); //ubtalent

                String str2=str.substring(0,cnt);//ih

                System.out.println(str1+str2);//ubtalentih

        }
}
```

Q)Write a java program to find out left rotation of a string?

input: str="ihubtalent"
cnt=2;

output: ntihubtale

```
class Test
{
        public static void main(String[] args)
        {
                String str="ihubtalent";
                int cnt=2;
                char[] carr=str.toCharArray();          // i h u b t a l e n t


                String str1=str.substring(0,carr.length-cnt);//ihubtale

                String str2=str.substring(carr.length-cnt,carr.length);//nt


                System.out.println(str2+str1);
        }
}
```

Q)Write a java program to display the string starting with vowels?

input: this is java class for you in a coming batch

output: is in a

```
class Test
{
        public static void main(String[] args)
        {
                String str="this is java class for you in a coming batch";

                String[] sarr=str.split(" ");

                String result="";
```

```
                for(String s:sarr)
                {
                        if(s.charAt(0)=='a'  ||  s.charAt(0)=='e'  ||  s.charAt(0)=='i'  ||
s.charAt(0)=='o' ||  s.charAt(0)=='u')
                        {
                                result+=s+" ";
                        }
                }
                System.out.println(result);
        }
}
```

Assignments
Q)Write a java program to display given string in below format?
input: ABBCCCDDDD
output: A1B2C3D4

Q)Write a java program to display permutation of string?
input: ABC
output: ABC
ACB
BAC
BCA
CAB
CBA

StringBuffer
If our content change frequently then it is never recommanded to use String object because for every change a new object will be created.
To overcome this limitation Sun Micro System introduced StringBuffer object.
In StringBuffer object all the required changes will be done in a single object only and it is mutable object.

constructor
1) StringBuffer sb=new StringBuffer()
It will create empty StringBuffer object with default initial capacity of 16.
If capacity reaches to maximum capacity then new capacity will be created with below formulea.
ex: new capacity = current_capacity+1*2;

```
class Test
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer();

                System.out.println(sb.capacity());//16
```

```
                sb.append("abcdefgjijklmnop");

                System.out.println(sb.capacity());//16

                sb.append("qr"); // 16+1*2=34

                System.out.println(sb.capacity());//34

        }
}
```

2) StringBuffer sb=new StringBuffer(int initialcapacity)
It will create StringBuffer with specified initial capacity.

```
class Test
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer(19);

                System.out.println(sb.capacity());//19


        }
}
```

3)StringBuffer sb=new StringBuffer(String str)
It will create StringBuffer object which is equivalent to String.
Here capacity will be created with below formulea.
ex: capacity = s.length+16;

```
class Test
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer("bhaskar");

                System.out.println(sb.capacity());//7+16=23
        }
}
```

Q)Write a java program to display reverse of a string?
input: hello
output: olleh

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

```
		String str="hello";

		String rev="";

		StringBuffer sb=new StringBuffer(str);

		rev=sb.reverse().toString();

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

Q)Write a java program to check given string is palindrome or not?

```
class Test
{
	public static void main(String[] args)
	{
		String str="madam";

		String rev="";

		StringBuffer sb=new StringBuffer(str);

		rev=sb.reverse().toString();

		if(str.equals(rev))
			System.out.println("It is palindrome string");
		else
			System.out.println("It is not palindrome string");
	}
}
```

StringBuilder

StringBuilder is exactly the same as StringBuffer with following differences.

StringBuffer	StringBuilder
All the methods present in StringBuffer are synchronized.	No method present in StringBuilder is synchronized.
At a time only one thread is allowed to execute.Hence we can achieve thread safety.	Multiple threads are allowed to execute.Hence we can't achieve thread safety.
Waiting time of a thread will increase effectively performance is low.	There is no waiting time of a thread effectively performance is high.
It is introduced in 1.0v.	It is introduced in 1.5v.

StringTokenizer

A StringTokenizer is a class which is present in java.util package.
It is used to tokenize the string irrespective of regular expression.
We can create StringTokenizer object as follow.

syntax: StringTokenizer st=new StringTokenizer(String,RegEx);

StringTokenizer class contains following methods.

ex:
```
public boolean hasMoreTokens()
public String nextToken()
public boolean hasMoreElements()
public Object nextElement()
public int countTokens()
```

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("this is java class");

		System.out.println(st.countTokens());//4
	}
}
```

*Note: Here default regular expression is space.

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("this is java class"," ");

		System.out.println(st.countTokens());//4
	}
}
```

ex:

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("this is java class"," ");

		while(st.hasMoreTokens())
		{
			String s=st.nextToken();
			System.out.println(s);
		}
```

```
	}
}
```

ex:

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("this is java class"," ");

		while(st.hasMoreElements())
		{
			String s=(String)st.nextElement();
			System.out.println(s);
		}
	}
}
```

ex:

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("9,99,999",",");

		while(st.hasMoreElements())
		{
			String s=(String)st.nextElement();
			System.out.println(s);
		}
	}
}
```

java.io package

File

```
	File f=new File("abc.txt");
```

File will check is there any abc.txt file already created or not.
If it is available it simply refers to that file.If it is not created then
it won't create any new file.

```
import java.io.*;
class Test
{
	public static void main(String[] args)
	{
```

```
                File f=new File("abc.txt");
                System.out.println(f.exists());//false
        }
}
```

A File object can be used to create a physical file.

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                File f=new File("abc.txt");
                System.out.println(f.exists());//false

                f.createNewFile();
                System.out.println(f.exists());//true

        }
}
```

A File object can be used to create a directory also.

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                File f=new File("bhaskar123");
                System.out.println(f.exists());//false

                f.mkdir();
                System.out.println(f.exists());//true

        }
}
```

Q)Write a java program to Create a "cricket123" folder and inside that folder create "abc.txt" file?

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                File f1=new File("cricket123");
                f1.mkdir();

                File f2=new File("cricket123","abc.txt");
```

```
                f2.createNewFile();

                System.out.println("Please check the location");

        }
}
```

FileWriter

FileWriter is used to write character oriented data into a file.

constructor

```
FileWriter fw=new FileWriter(String s);
FileWriter fw=new FileWriter(File f);
ex:     FileWriter fw=new FileWriter("aaa.txt");
                        or
        File f=new File("aaa.txt");
        FileWriter fw=new FileWriter(f);
```

If file does not exist then FileWriter will create a physical file.

Methods

1. write(int ch) It will insert single character into a file.
2. write(char[] ch) It will insert array of characters into a file.
3. write(String s) It will insert String into a file.
4. flush() It gives guaranttee that last character of a file is also inserted.
5. close() It is used to close the FileWriter object.

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                FileWriter fw=new FileWriter("aaa.txt");

                fw.write(98);// b
                fw.write("\n");

                char[] ch={'a','b','c'};
                fw.write(ch);
                fw.write("\n");

                fw.write("bhaskar\nsolution");
                fw.flush();
                fw.close();
                System.out.println("Please check the location");
        }
}
```

FileReader

It is used to read character oriented data from a file.

constructor

FileReader fr=new FileReader(String s);

FileReader fr=new FileReader(File f);

Eg: FileReader fr=new FileReader("aaa.txt");

or

File f=new File("aaa.txt");

FileReader fr=new FileReader(f);

Methods

1. read() It will read next character from a file and return unicode value. If next character is not available then it will return -1.
2. read(char[] ch) It will read collection of characters from a file.
3. close() It is used to close FileReader object.

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		FileReader fr=new FileReader("aaa.txt");

		int i=fr.read();
		while(i!=-1)
		{
			System.out.print((char)i);
			i=fr.read();
		}
		fr.close();
	}
}
```

ex:2

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		FileReader fr=new FileReader("aaa.txt");

		char[] carr=new char[255];

		//load the data from file to char array
		fr.read(carr);

		//reading the data from char array
		for(char c:carr)
```

```
            {
                System.out.print(c);
            }

            fr.close();

        }
}
```

Usage of FileWriter and FileReader is not recommanded to use

While inserting the data by using FileWriter ,we need to insert line seperator(\n) which is very headache for the programmer.

While reading the data by using FileReader object ,we need to read character by character which is not convenient to the programmer.

To overcome this limitation Sun micro system introduced BufferedWriter and BufferedReader.

BufferedWriter

It is used to insert character oriented data into a file.

constructor

```
BufferedWriter bw=new BufferedWriter(Writer w);
BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);
```

BufferedWriter object does not communicate with files directly.

It will take the support of some writer objects.

```
ex:     FileWriter fw=new FileWriter("bbb.txt");
        BufferedWriter bw=new BufferedWriter(fw);
                            or
        BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));
```

Methods

1. write(int ch)	It will insert single character into a file.
2. write(char[] ch)	It will insert array of characters into a file.
3. write(String s)	It will insert String into a file.
4. flush()	It gives guaranttee that last character of a file is also inserted.
5. close()	It is used to close the BufferedWriter object.
6. newLine()	It will insert new line into a file.

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {

        BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));
                bw.write(98);//b
                bw.newLine();
                char[] ch={'a','b','c'};
                bw.write(ch);
```

```
                bw.newLine();

                bw.write("bhaskar");
                bw.newLine();

                bw.flush();
                bw.close();
                System.out.println("Please check the location");
        }
}
```

BufferedReader

It is enhanced reader to read character oriented data from a file.

constructor

BufferedReader br=new BufferedReader(Reader r);

BufferedReader br=new BufferedReader(Reader r,int buffersize);

BufferedReader object can't communicate with files directly.IT will take support of some reader objects.

ex: FileReader fr=new FileReader("bbb.txt");
BufferedReader br=new BufferedReader(fr);
or
BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));

The main advantage of BufferedReader over FileReader is we can read character line by line instead of character by character.

methods

1. read()	It will read next character from a file and return unicode value. If next character is not available then it will return -1.
2. read(char[] ch)	It will read collection of characters from a file.
3. close()	It is used to close BufferedReader object.
4. nextLine()	It is used to read next line from the file.If next line is not then it available will return null.

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));
                String line=br.readLine();
                while(line!=null)
                {
                        System.out.println(line);
                        line=br.readLine();
                }
                br.close();
        }
}
```

PrintWriter

It is enhanced write to write character oriented data into a file.

constructor

PrintWriter pw=new PrintWriter(String s);
PrintWriter pw=new PrintWriter(File f);
PrintWriter pw=new PrintWriter(Writer w);

PrintWriter can communicate with files directly and it will take the support of some writer objects.

ex: PrintWriter pw=new PrintWriter("ccc.txt");
or
PrintWriter pw=new PrintWriter(new File("ccc.txt"));
or
PrintWriter pw=new PrintWriter(new FileWriter("ccc.txt"));

The main advantage of PrintWriter over FileWriter and BufferedWriter is we can insert any type of data.

Assume if we want insert primitive values then PrintWriter is best choice.

methods

write(int ch)
write(char[] ch)
write(String s)
flush()
close()

writeln(int i)
writeln(float f)
writeln(double d)
writeln(String s)
writeln(char c)
writeln(boolean b

write(int i)
write(float f)
write(double d)
write(String s)
write(char c)
write(boolean b)

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		PrintWriter pw=new PrintWriter("ccc.txt");

		pw.write(100);// d
```

```
                pw.println(100);// 100
                pw.print('a');
                pw.println(true);
                pw.println("hi");
                pw.println(10.5d);

                pw.flush();
                pw.close();
                System.out.println("Please check the location");
        }
}
```

Various ways to provide input values from keyboard

There are many ways to provide input values from keyboard.

1) command line argument
2) Console class
3) BufferedReader class
4) Scanner class

1) command line argument

```
class Test
{
        public static void main(String[] args)
        {
                String name=args[0];

                System.out.println("Welcome :"+name);
        }
}
```

o/p:

```
        javac   Test.java

        java    Test   DennisRitchie
```

2) Console class

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                Console c=System.console();

                System.out.println("Enter the name :");

                String name=c.readLine();
```

```
		System.out.println("Welcome :"+name);
	}
}
```

3) BufferedReader class

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

		System.out.println("Enter the name :");

		String name=br.readLine();

		System.out.println("Welcome :"+name);
	}
}
```

4) Scanner class

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the employee id :");
		int id=sc.nextInt();

		System.out.println("Enter the employee name :");
		String name=sc.next();

		System.out.println("Enter the employee salary :");
		float sal=sc.nextFloat();

		System.out.println(id+" "+name+" "+sal);
	}
}
```

Exception Handling

Q)What is the difference between Exception and Error?

Exception

Exception is a problem for which we can provide solution programmatically.

Exception will raise due to syntax error.

ex: FileNotFoundException
ArithmeticException
IllegalArgumentException

Error

Error is a problem for which we can't provide solution programmatically.

Error will raise due to lack of system resources.

ex: OutOfMemoryError
LinkageError
StackOverFlowError
and etc

As a part of java application development, it is a responsibility of a programmer to provide smooth termination for every java program.

We have two types of terminations.

1)Smooth termination / graceful termination

2)Abnormal termination

1)Smooth termination

During the program execution suppose if we are not getting any interruption in the middle of the program such type of termination is called smooth termination.

2)Abnormal termination

During the program execution suppose if we are getting any interruption in the middle of the program such type of termination is called abnormal termination

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(10/0);
	}
}
```

If any exception raised in our program , we must and should handle that exception otherwise our program will terminates abnormally.

Exception will display name of the exception ,description of the exception and line number of the exception.

Exception

It is a unwanted, unexpected event which disturbs normal flow of our program.

Exceptions always raised at runtime so they are also known as runtime events.

The main objective of exception handing is to provide graceful termination.

In java, exceptions are divided into two types.

1)Predefined exception

2)Userdefined exceptions

1)Predefined exception
Built-In exceptions are called predefined exceptions.
It classified into two types.
i)Checked exceptions
ii)Unchecked exceptions

i)Checked exceptions
Exceptions which are checked by the compiler at the time of compilation are called checked exceptions.
ex: InterruptException
IOException
EOFException
and etc.
ii)Unchecked exceptions
Exceptions which are checked by the JVM at the time of runtime are called unchecked exceptions.
ex: ArithmeticException
IllegalArgumentException
ClassCastException
and etc.

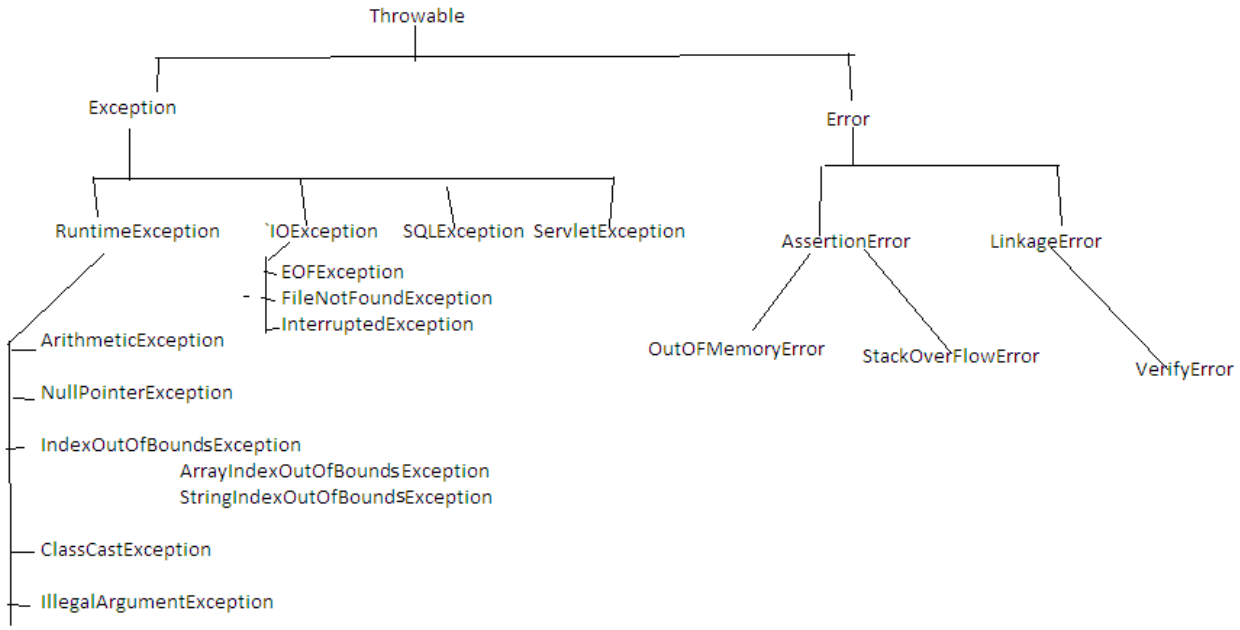


Diagram: java32.1

If any checked exception raised in our program , we must and should handle that exception by using try and catch block.

try block
It is a block which contains risky code.
A try block associate with catch block.

If any exception raised in try block then it won't be executed.
A try block is used to throw the exception to catch block.

catch block
It is a block which contains Error Handling Code.
A catch block always associate with try block.
A catch block will take exception name as a parameter and that name must match with exception class name.
If there is no exception in try block then catch block won't be executed.
A catch block is used to catch the exception which is thrown by try block.

```
syntax: try
        {
                -
                - //Risky Code
                -
        }
        catch(ArithmeticException ae)
        {
                -
                - //Error Handling Code
                -
        }
```

```
class Test
{
        public static void main(String[] args)
        {
                try
                {
                        System.out.println("try-block");
                }
                catch (Exception e)
                {
                        System.out.println("catch-block");
                }
        }
}
```

o/p: try-block

ex:

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

```
                System.out.println(10/0);
            }
            catch (ArithmeticException ae)
            {
                System.out.println("catch-block");
            }
        }
}
```
o/p: catch-block

ex:

```
class Test
{
        public static void main(String[] args)
        {
            try
            {
                System.out.println("stmt1");
                System.out.println(10/0);
                System.out.println("stmt2");
            }
            catch (ArithmeticException ae)
            {
                System.out.println("catch-block");
            }
        }
}
```
o/p: stmt1
catch-block

A try with multiple catch blocks

A try block can have multiple catch blocks.

If a try block contains multiple catch block then order of catch block is very important ,it should be from child to parent but not from parent to child.

```
class Test
{
        public static void main(String[] args)
        {
            try
            {
                System.out.println(10/0);
            }
            catch (ArithmeticException ae)
            {
                System.out.println("From AE");
            }
```

```
                catch (RuntimeException re)
                {
                        System.out.println("From RE");
                }
                catch (Exception e)
                {
                        System.out.println("From E");
                }
        }
}
```

Various ways to display exception details

Throwable class defined following three methods to display exception details.

1) printStackTrace()

It will display name of the exception, description of the exception and line number of the exception.

2) toString()

It will display name of the exception and description of the exception.

3) getMessage()

It will display description of the exception.

```
class Test
{
        public static void main(String[] args)
        {
                try
                {
                        System.out.println(10/0);
                }
                catch (ArithmeticException ae)
                {
                        ae.printStackTrace();

                        System.out.println("============================");

                        System.out.println(ae.toString());

                        System.out.println("============================");

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

A single catch block can handle multiple exceptions.

ex:

```
class Test
```

```
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println(10/0);
		}
		catch (ArithmeticException | ClassCastException | IllegalArgumentException e)
		{
			e.printStackTrace();
		}
	}
}
```

finally block

It is never recommanded to maintain cleanup code in try block because if any exception raised in try block then it won't be executed.

It is never recommanded to maintain cleanup code in catch block because if no exception raised in try block then catch block won't be executed.

We need a place where we can maintain cleanup code and it will execute irrespective of exception raised or not.Such block is called finally block.

A finally block always associate with try and catch block.

syntax:

```
try
	{
		-
		- //Risky Code
		-
	}
	catch(ArithmeticException ae)
	{
		-
		- //Error Handling Code
		-
	}
	finally
	{
		-
		- //Cleanup Code
		-
	}
```

ex:

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

```
		{
			System.out.println("try-block");
		}
		catch (ArithmeticException ae)
		{
			ae.printStackTrace();
		}
		finally
		{
			System.out.println("finally-block");
		}
	}
}
```
o/p: try-block
finally-block

ex:

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println(10/0);
		}
		catch (ArithmeticException ae)
		{
			ae.printStackTrace();
		}
		finally
		{
			System.out.println("finally-block");
		}
	}
}
```
o/p: java.lang.ArithmeticException: / by zero
at Test.main(Test.java:7)
finally-block

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)
	{
		FileWriter fw=null;

		try
```

```
		{
			fw=new FileWriter("xyz.txt");
			fw.write(98);//b
			fw.write("\n");
			char[] carr={'a','b','c'};
			fw.write(carr);
			fw.flush();
		}
		catch (IOException ioe)
		{
			ioe.printStackTrace();
		}
		finally
		{
			System.out.println("Please check the location");

			try
			{
				fw.close();
			}
			catch (IOException ioe)
			{
				ioe.printStackTrace();
			}
		}
	}
}
```

A try with finally combination is valid in java.

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println("try-block");
		}
		finally
		{
			System.out.println("finally-block");
		}
	}
}
```

Q)What is the difference between final, finally and finalized method?

final

A final is a modifier which is applicable for variables,methods and classes.
If we declare any variable as final then reassignment of that variable is not possible.
If we declare any method as final then overriding of that method is not possible.
If we declare any class as final then creating child class is not possible.

finally

It is a block which contains cleanup code and it will execute irrespective of exception raised or not.

finalized method

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw statement

Sometimes we will create exception object explicitly and handover to JVM manually by using throw statement.

ex: throw new ArithmeticException("Don't divide by zeroo");

```
class Test
{
	public static void main(String[] args)
	{
		throw new ArithmeticException("don't divide by zero");
	}
}
```

throws statement

If any checked exception raised in our program so we must and should handle that exception by using try and catch block or by using throws statement.

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			Thread.sleep(3000);
			System.out.println("Welcome to Java");
		}
		catch (InterruptedException ie)
		{
			ie.printStackTrace();
		}

	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)throws InterruptedException
	{

			Thread.sleep(5000);
			System.out.println("Welcome to Java");
	}
}
```

2)Userdefined exceptions

Exceptions which are created by the user based on the application requirement are called userdefined exceptions.

ex: NoInterestInPracticeException
EnjoyingClassesWithMobileException
ACNotWorkingException
TooYoungException
TooOldException
and etc.

```
import java.util.Scanner;
class TooYoungException extends RuntimeException
{
	TooYoungException(String s)
	{
		super(s);
	}
}
class TooOldException extends RuntimeException
{
	TooOldException(String s)
	{
		super(s);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the age :");
		int age=sc.nextInt();
		if(age<18)
		throw new TooYoungException("U r not eligible to vote");
		else
```

```
                throw new TooOldException("U r eligible to vote");
        }
}
```

Generics

Arrays are typesafe.

It means we can provide guarantee that what type of elements are present in arrays.

If requirement is there to store string values then it is recommanded to use String[] array.

```
ex:     String[] sarr=new String[10];
        sarr[0]="hi";
        sarr[1]="hello";
        sarr[2]="bye";
        sarr[3]=10;   --> invalid
```

At the time of retrieving the data from array we don't need to perform any typecasting.

```
ex:     String[] sarr=new String[10];
        sarr[0]="hi";
        sarr[1]="hello";
        sarr[2]="bye";
        -
        -
        String val=sarr[0];
```

Collections are not typesafe.

We can't give guarantee that what type of elements are present in Collections.

If requirement is there to store String values then it is never recommanded to use ArrayList because we won't get any compile time error or runtime error but sometimes our program will get failure.

```
ex:     ArrayList al=new ArrayList();
        al.add("hi");
        al.add("hello");
        al.add("bye");
        al.add(10);
```

At the time of retrieving the data from Collections ,compulsary we need to perform typecasting.

```
ex:     ArrayList al=new ArrayList();
        al.add("hi");
        al.add("hello");
        al.add("bye");
        al.add(10);
        -
        -
        String val=(String)al.get(0);
```

To overcome above limitations Sun Micro System introduced Generics concept in 1.5 version.

The main objective of generics are.

1) To make Collections as typesafe.

2) To avoid typecasting problem.

java.util package

Q)What is the difference between Arrays and Collections ?

Arrays	Collections
It is a collection of homogeneous data elements.	It is a collection of homogeneous and hetrogeneous data elements.
Arrays are fixed in size.	Collections are growable in nature.
Performance point of view arrays are recommanded to use.	Memory point of view Collections are recommanded to use.
It is typesafe.	It is not typesafe.
Arrays can hold primitive types and object types.	Collections can hold only object types
Arrays are not implemented based on data structure concept so we can't expect any ready made methods.For every logic we need to write the code explicitly.	Collections are implemented based on data structure concept so we can expect ready made methods.

Collection

Collection is an interface which is present in java.util package.

It is a root interface for entire Collection framework.

If we want to represent group of individual objects in a single entity then we need to use Collection interface.

Collection interface contains following methods which are applicable for entire Collection objects.

ex: cmd> javap java.util.Collection

```
  public abstract int size();
  public abstract boolean isEmpty();
  public abstract boolean contains(java.lang.Object);
  public abstract java.util.Iterator<E> iterator();
  public abstract java.lang.Object[] toArray();
  public abstract <T> T[] toArray(T[]);
  public abstract boolean add(E);
  public abstract boolean remove(java.lang.Object);
  public abstract boolean containsAll(java.util.Collection<?>);
  public abstract boolean addAll(java.util.Collection<? extends E>);
  public abstract boolean removeAll(java.util.Collection<?>);
  public boolean removeIf(java.util.function.Predicate<? super E>);
  public abstract boolean retainAll(java.util.Collection<?>);
  public abstract void clear();
  public abstract boolean equals(java.lang.Object);
  public abstract int hashCode();
  public java.util.Spliterator<E> spliterator();
  public java.util.stream.Stream<E> stream();
  public java.util.stream.Stream<E> parallelStream();
```

List

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are allowed and order is preserved then we need to use List interface.

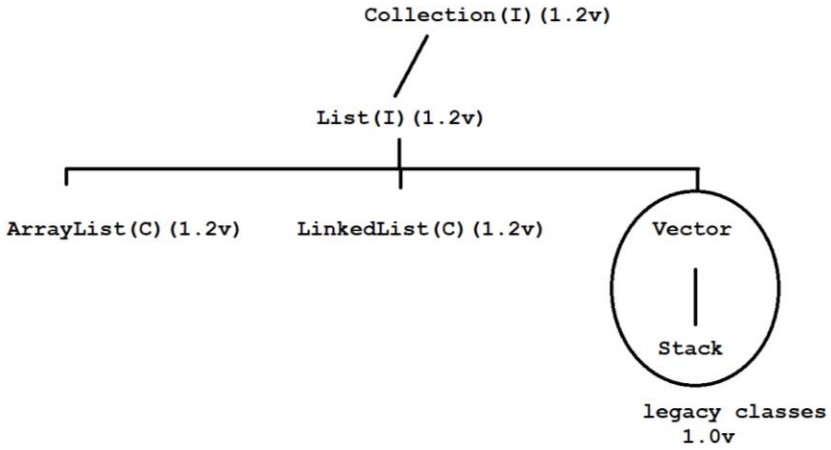


Diagram : java33.1

ArrayList

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and RandomAccess interface.

If our frequent operation is a retrieval operation then ArrayList is best choice.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("one");
		al.add("two");
		al.add("three");
		System.out.println(al);//[one,two,three]
		al.add("one");
		System.out.println(al);//[one,two,three,one]
		al.add(10);
		System.out.println(al);//[one,two,three,one,10]
		al.add(null);
		System.out.println(al);//[one,two,three,one,10,null]
	}
}
```

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList<String> al=new ArrayList<String>();
		al.add("one");
		al.add("two");
		al.add("three");
		System.out.println(al);//[one,two,three]
		al.add("one");
		System.out.println(al);//[one,two,three,one]
		al.add(null);
		System.out.println(al);//[one,two,three,one,null]
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList<String> al=new ArrayList<String>();
		al.add("one");
		al.add("two");
		al.add("three");

		for(int i=0;i<al.size();i++)
		{
			String s=al.get(i);
			System.out.println(s);
		}
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList<String> al=new ArrayList<String>();
		al.add("one");
		al.add("two");
		al.add("three");
```

```
                System.out.println(al.isEmpty());//false

                System.out.println(al.contains("one"));//true

                al.remove("one");
                System.out.println(al);//[two,three]

                al.clear();
                System.out.println(al);//[]
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                List<String> list=new ArrayList<String>();
                list.add("one");
                list.add("two");
                list.add("three");
                System.out.println(list);//[one,two,three]
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                List<String> list=new ArrayList<String>();
                list.add(new String("one"));
                list.add(new String("two"));
                list.add(new String("three"));
                System.out.println(list);//[one,two,three]
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
```

```
	{
		List<String> l=Arrays.asList("one","two","three","four");
		System.out.println(l);
	}
}
```

LinkedList

The underlying data structure is double linkedlist.
Duplicate objects are allowed.
Order is preserved.
Hetrogeneous objects are allowed.
Null insertion is possible.
It implements Serializable, Cloneable and Deque interface.
If our frequent operation is a adding and removing in the middle then LinkedList is a best choice.
LinkedList contains following methods.

ex: addFirst()
addLast()
getFirst()
getLast()
removeFirst()
removeLast()
and etc.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		LinkedList ll=new LinkedList();
		ll.add("one");
		ll.add("two");
		ll.add("three");
		System.out.println(ll);//[one,two,three]
		ll.add("one");
		System.out.println(ll);//[one,two,three,one]
		ll.add(10);
		System.out.println(ll);//[one,two,three,one,10]
		ll.add(null);
		System.out.println(ll);//[one,two,three,one,10,null]
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
```

```
	{
		LinkedList<String> ll=new LinkedList<String>();
		ll.add("one");
		ll.add("two");
		ll.add("three");
		System.out.println(ll);//[one,two,three]
		ll.addFirst("gogo");
		ll.addLast("jojo");
		System.out.println(ll);//[gogo,one,two,three,jojo]

		System.out.println(ll.getFirst());//gogo
		System.out.println(ll.getLast());//jojo

		ll.removeFirst();
		ll.removeLast();
		System.out.println(ll);//[one,two,three]
	}
}

ex:
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		LinkedList<String> ll1=new LinkedList<String>();
		ll1.add("one");
		ll1.add("two");
		ll1.add("three");
		System.out.println(ll1);//[one,two,three]

		LinkedList<String> ll2=new LinkedList<String>();
		ll2.add("raja");
		System.out.println(ll2);//[raja]

		ll2.addAll(ll1);
		System.out.println(ll2);//[raja,one,two,three]

		System.out.println(ll2.containsAll(ll1));// true

		ll2.removeAll(ll1);
		System.out.println(ll2);//[raja]
	}
}
```

Vector

The underlying data structure is resizable array or growable array.
Duplicate objects are allowed.
Insertion order is preserved.
Hetrogeneous objects are allowed.
Null insertion is possible.
It implements Serializable, Cloneable and RandomAccess interface.
All the methods present in Vector are synchronized.Hence we can achieve thread safety.
Vector contains following methods.

ex: addElement()
firstElement()
lastElement()
removeElementAt()
removeAllElements()
and etc.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Vector v=new Vector();
		v.add("one");
		v.add("two");
		v.add("three");
		System.out.println(v);//[one,two,three]
		v.add("one");
		System.out.println(v);//[one,two,three,one]
		v.add(10);
		System.out.println(v);//[one,two,three,one,10]
		v.add(null);
		System.out.println(v);//[one,two,three,one,10,null]

	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Vector v=new Vector();
		v.addElement("one");
		v.addElement("two");
		v.addElement("three");
		System.out.println(v);//[one,two,three]
```

```
                System.out.println(v.firstElement());//one
                System.out.println(v.lastElement());//three

                v.removeElementAt(1);
                System.out.println(v);//[one,three]

                v.removeAllElements();
                System.out.println(v);//[]

        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                Vector v=new Vector();
                v.add("one");
                v.add("two");
                v.add("three");
                System.out.println(v);//[one,two,three]

                System.out.println(v.get(0));//one
                System.out.println(v.get(v.size()-1));//three

                v.remove("two");
                System.out.println(v);//[one,three]

                v.clear();
                System.out.println(v);//[]

        }
}
```

Stack

It is a child class of Vector class.
If we depends upon Last In First Out (LIFO) order then we need to use Stack.
We can create Stack object as follow.
ex: Stack s =new Stack();

Methods

1) push(Object o) It will insert the element to stack.
2) pop() It will remove the element from stack.
3) peek() It will return toppest element of stack.

4) isEmpty() It is used to check stack is empty or not.
5) search(Object o) It will return offset value if element is found otherwiser it will return -1.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Stack<String> s=new Stack<String>();
		s.push("A");
		s.push("B");
		s.push("C");
		System.out.println(s);//[A,B,C]

		s.pop();
		System.out.println(s);//[A,B]

		System.out.println(s.peek());//B

		System.out.println(s.isEmpty());//false

		System.out.println(s.search("Z")); // -1

		System.out.println(s.search("A"));//2
	}
}
```

Set

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are not allowed and order is not preserved then we need to use Set interface.

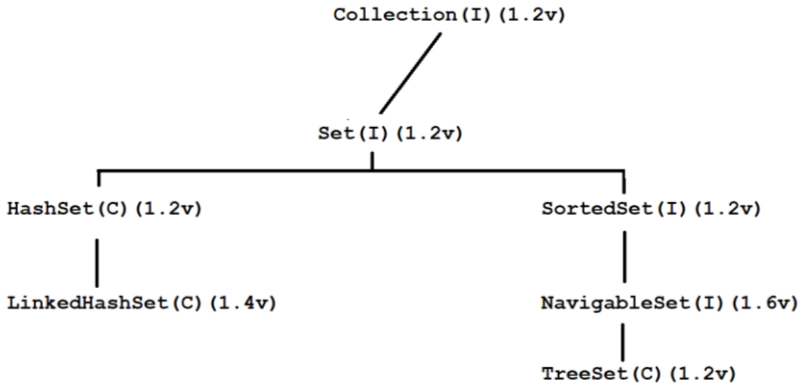


Diagram: java34.1

HashSet

The underlying data structure is Hashtable.

Duplicate objects are not allowed.

Order is not preserved because it will take hash code of an object.
Hetrogeneous objects are allowed.
Null insertion is possible.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		HashSet hs=new HashSet();
		hs.add("one");
		hs.add("nine");
		hs.add("five");
		hs.add("eight");
		System.out.println(hs);//[nine, one, five, eight]
		hs.add("one");
		System.out.println(hs);//[nine,one,five,eight]
		hs.add(10);
		System.out.println(hs);//[nine, one, 10, five, eight]
		hs.add(null);
		System.out.println(hs);//[null, nine, one, 10, five, eight]
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		HashSet<String> hs=new HashSet<String>();
		hs.add("one");
		hs.add("nine");
		hs.add("five");
		hs.add("eight");
		System.out.println(hs);//[nine, one, five, eight]
		hs.add("one");
		System.out.println(hs);//[nine,one,five,eight]
		hs.add(null);
		System.out.println(hs);//[null, nine, one, five, eight]
	}
}
```

LinkedHashSet

LinkedHashSet is a child class of HashSet class.
LinkedHashSet is exactly same as HashSet class with following differences.

HashSet	LinkedHashSet
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
It is introduced in 1.2v.	It is introduced in 1.4v.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		LinkedHashSet<String> lhs=new LinkedHashSet<String>();
		lhs.add("one");
		lhs.add("nine");
		lhs.add("five");
		lhs.add("eight");
		System.out.println(lhs);//[one,nine,five,eight]
		lhs.add("one");
		System.out.println(lhs);//[one,nine,five,eight]
		lhs.add(null);
		System.out.println(lhs);//[one,nine,five,eight,null]
	}
}
```

TreeSet

The underlying data structure is Balanced Tree.
Duplicate objects are not allowed.
Insertion order is not preserved because it will take sorting order of an object.
Hetrogenous objects are not allowed if we insert then we will get ClassCastException.
For empty TreeSet if we trying to insert null then we will get NullPointerException.
After inserting the elements if we are trying to insert null then we will get NullPointerException.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeSet ts=new TreeSet();
		ts.add(10);
		ts.add(1);
		ts.add(5);
		ts.add(7);
		ts.add(3);
		System.out.println(ts);//[1,3,5,7,10]
		ts.add(10);
		System.out.println(ts);//[1,3,5,7,10]
		//ts.add("hi");
```

```
                //System.out.println(ts);//R.E ClassCastException
                //ts.add(null);
                //System.out.println(ts);// R.E NullPointerException

        }
}
```

Q)What is the difference between Comparable and Comparator interface.

Comparable

Comparable interface present in java.lang package.
It contains only one method i.e compareTo() method.
ex: compareTo(Object obj1,Object obj2);
It will return -ve if obj1 comes before obj2.
It will return +ve if obj1 comes after obj2.
it will return 0 if both objects are same.
If we depend upon default natural sorting then we need to use Comparable interface.

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("A".compareTo("Z"));     //-25

                System.out.println("Z".compareTo("A")); // 25

                System.out.println("K".compareTo("K")); // 0

        }
}
```

Comparator

Comparator is an interface which is present in java.util package.
Comparator interface contains following two methods. i.e equals() and compare() method.
ex: public abstract int compare(Object obj1, Object obj2);
It will return +ve if obj1 comes before obj2.
It will return -ve if obj1 comes after obj2.
It will return 0 if both objects are same.
public abstract boolean equals(java.lang.Object);
Implementation of equals() method is optional because it is present in Object class so it is available through inheritance.
Implementation of compare() method is mandatory.
If we depends upon customized sorting order then we need to use Comparator interface.

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());
```

```
                ts.add(10);
                ts.add(1);
                ts.add(5);
                ts.add(7);
                ts.add(3);
                System.out.println(ts);//10 7 5 3 1
        }
}
class MyComparator implements Comparator
{
        public int compare(Object obj1,Object obj2)
        {
                Integer i1=(Integer)obj1;
                Integer i2=(Integer)obj2;
                if(i1<i2)
                        return 1;
                else if(i1>i2)
                        return -1;
                else
                        return 0;
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());
                ts.add(10);
                ts.add(1);
                ts.add(5);
                ts.add(7);
                ts.add(3);
                System.out.println(ts);//1 3 5 7 10
        }
}
class MyComparator implements Comparator
{
        public int compare(Object obj1,Object obj2)
        {
                Integer i1=(Integer)obj1;
                Integer i2=(Integer)obj2;
                if(i1<i2)
                        return -1;
```

```
            else if(i1>i2)
                return 1;
            else
                return 0;
        }
}
```

Map

It is not a child interface of Collection interface.

If we want to represent group of individual objects in key,value pair then we need to use Map interface.

Key can't be duplicated but value can be duplicated.

Here key and value both must be objects.

Each key and value pair is called one entry.

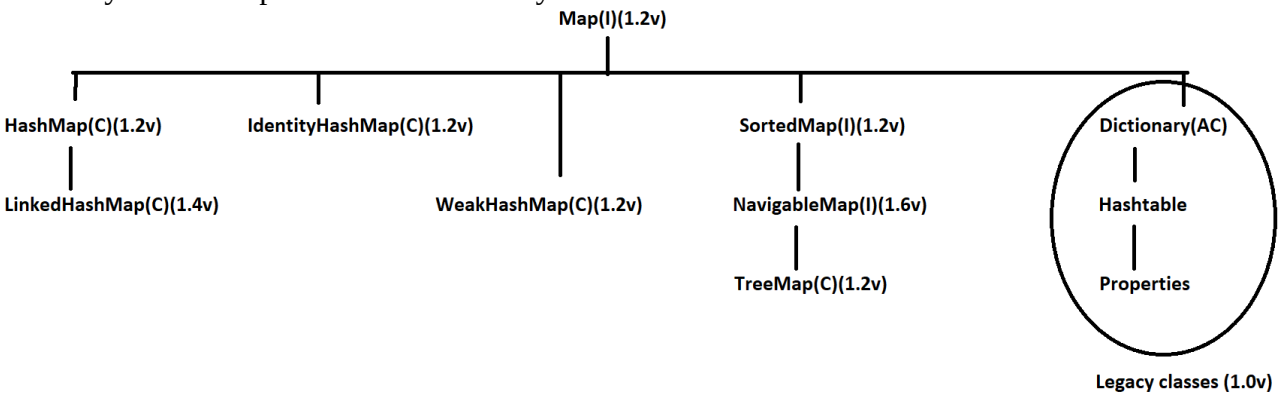


Diagram: java35.1

HashMap

The underlying data structure is Hashtable.

Key can't be duplicated but value can be duplicated.

Insertion order is not preserved because it will take hash code of the key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is possible for both key and value.

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        HashMap<String,String> hm=new HashMap<String,String>();
        hm.put("one","raja");
        hm.put("nine","nelson");
        hm.put("six","jose");
        hm.put("eight","Erick");
        System.out.println(hm);//{nine=nelson, six=jose, one=raja, eight=Erick}
        hm.put("one","jojo");
        System.out.println(hm);//{nine=nelson, six=jose, one=jojo, eight=Erick}
        hm.put(null,null);
```

```
		System.out.println(hm);//{null=null, nine=nelson, six=jose, one=jojo, eight=Erick}
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		HashMap<String,String> hm=new HashMap<String,String>();
		hm.put("one","raja");
		hm.put("nine","nelson");
		hm.put("six","jose");
		hm.put("eight","Erick");

		Set s=hm.keySet();
		System.out.println(s);//[nine, six, one, eight]

		Collection c=hm.values();
		System.out.println(c);//[nelson, jose, raja, Erick]

		Set s1=hm.entrySet();
		System.out.println(s1);//[nine=nelson, six=jose, one=raja, eight=Erick]
	}
}
```

LinkedHashMap

It is a child class of HashMap class.

LinkedHashMap is exactly same as HashMap class with following differences.

HashMap	LinkedHashMap
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
Introduced in 1.2v.	Introduced in 1.4v.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		LinkedHashMap<String,String> lhm=new LinkedHashMap<String,String>();
		lhm.put("one","raja");
		lhm.put("nine","nelson");
		lhm.put("six","jose");
```

```
		lhm.put("eight","Erick");
		System.out.println(lhm);//{one=raja, nine=nelson, six=jose, eight=Erick}

	}
}
```

TreeMap

The underlying datastructure is RED BLACK TREE.
Key can't be duplicated but value can be duplicated.
Insertion order is not preserved because it will take sorting order of the key.
If we depend upon default natural sorting order then key can be homogeneous and comparable.
If we depend upon customized sorting order then key can be hetrogeneous and non-comparable.
Key can't be null but value can be null.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeMap<Integer,String> tm=new TreeMap<Integer,String>();
		tm.put(10,"ten");
		tm.put(1,"one");
		tm.put(5,"five");
		tm.put(7,"seven");
		tm.put(3,"three");
		System.out.println(tm);//{1=one, 3=three, 5=five, 7=seven, 10=ten}
		tm.put(4,null);
		System.out.println(tm);//{1=one, 3=three, 4=null, 5=five, 7=seven, 10=ten}
		//tm.put(null,"nine");
		//System.out.println(tm);//R.E NullPointerException

	}
}
```

Hashtable

The underlying datastructure is Hashtable.
Key can't be duplicated but value can be duplicated.
Insertion order is not preserved because it will take descending order of key.
Hetrogeneous objects are allowed for both key and value.
Key and value can't be null.

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Hashtable<Integer,String> ht=new Hashtable<Integer,String>();
		ht.put(10,"ten");
```

```
                ht.put(1,"one");
                ht.put(5,"five");
                ht.put(7,"seven");
                ht.put(3,"three");
                System.out.println(ht);//{10=ten, 7=seven, 5=five, 3=three, 1=one}
                //ht.put(4,null);
                //System.out.println(ht);//R.E NullPointerException
                //ht.put(null,"four");
                //System.out.println(ht);//R.E NullPointerException

        }
}
```

Types of Cursors

Cursor is used to read objects one by one from Collections.

We have three types of cursors in Java.

1) Enumeration
2) Iterator
3) ListIterator

1)Enumeration

It is used to read objects one by one from Legacy Collection objects.

We can create Enumeration object as follow.

ex: Enumeration e=v.elements();

Enumeration interface contains following two methods.

ex: public boolean hasMoreElements();
    public Object nextElement();

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                Vector v=new Vector();
                for(int i=1;i<=10;i++)
                {
                        v.add(i);
                }
                System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

                Enumeration e=v.elements();
                while(e.hasMoreElements())
                {
                        Integer i=(Integer)e.nextElement();
                        System.out.println(i);
                }
        }
}
```

Limitations with Enumeration

Using Enumeration we can read objects one by one only from legacy Collection objects.Hence it is not a universal cursor.

Using Enumeration interface we can perform read operation but not remove operation.

To overcome this limitation we need to use Iterator interface.

2)Iterator

It is used to read objects one by one from any Collection object.Hence it is a universal cursor.

Using Iterator interface we can perform read and remove operations.

We can create Iterator object as follow.

ex: Iterator itr=al.iterator();

Iterator interface contains following three methods.

ex: public boolean hasNext()
public Object next()
public void remove()

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		for(int i=1;i<=10;i++)
		{
			al.add(i);
		}
		System.out.println(al);//[1,2,3,4,5,6,7,8,9,10]

		Iterator itr=al.iterator();
		while(itr.hasNext())
		{
			Integer i=(Integer)itr.next();
			if(i%2==0)
			{
				itr.remove();
			}
		}
		System.out.println(al);//[1,3,5,7,9]

	}
}
```

Limitations with Iterator

> Using Enumeration and Iterator we can read objects only in forward direction but not in backward direction.Hence they are not bi-directional cursors.

> Using Iterator interface we can perform read and remove operation but not adding and replacement of new objects.

> To overcome this limitation we need to use ListIterator interface.

3)ListIterator

It is a child interface of Iterator interface.

ListIterator is used to read objects only from List Collection objects.Hence it is a bi-directional cursor.

Using ListIterator we can perform read, remove, adding and replacement of new objects.

We can create ListIterator object as follow.

ex: ListIterator litr=al.listIterator();

ListIterator interface contains following nine methods.

ex: public boolean hasNext()
public Object next()
public boolean hasPrevious()
public Object previous()
public void add(Object o)
public void set(Object o)
public int nextIndex()
public int previousIndex()

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("chiru");
		al.add("bala");
		al.add("nag");
		System.out.println(al);//[venki,chiru,bala,nag]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			System.out.println(s);
		}

	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
```

```
                al.add("chiru");
                al.add("bala");
                al.add("nag");
                System.out.println(al);//[venki,chiru,bala,nag]

                ListIterator litr=al.listIterator();
                while(litr.hasNext())
                {
                        String s=(String)litr.next();
                        if(s.equals("bala"))
                        {
                                litr.remove();
                        }
                }
                System.out.println(al);//[venki,chiru,nag]

        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                ArrayList al=new ArrayList();
                al.add("venki");
                al.add("chiru");
                al.add("bala");
                al.add("nag");
                System.out.println(al);//[venki,chiru,bala,nag]

                ListIterator litr=al.listIterator();
                while(litr.hasNext())
                {
                        String s=(String)litr.next();
                        if(s.equals("bala"))
                        {
                                litr.add("pavan");
                        }
                }
                System.out.println(al);//[venki, chiru, bala, pavan, nag]

        }
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("chiru");
		al.add("bala");
		al.add("nag");
		System.out.println(al);//[venki,chiru,bala,nag]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("bala"))
			{
				litr.set("Allu");
			}
		}
		System.out.println(al);//[venki, chiru, Allu, nag]

	}
}
```

Interview Questions

Q)Write a java program to find out repeating words in a given string?

input: this is is java java class

output: this=1 , is=2 , java=2 , class=1

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		String str="this is is java java class";

		LinkedHashMap<String,Integer> lhm=new LinkedHashMap<String,Integer>();

		String[] sarr=str.split(" ");

		for(String s:sarr)
		{
			if(lhm.get(s)!=null)
			{
```

```
                    lhm.put(s,lhm.get(s)+1);
                }
                else
                {
                    lhm.put(s,1);
                }
            }
            System.out.println(lhm);
        }
}
```

Q)Write a java program to find out repeating alphabets in a given string?
input: java
output: j=1,a=2,v=1

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        String str="java";

        LinkedHashMap<Character,Integer> lhm=new
LinkedHashMap<Character,Integer>();

        char[] carr=str.toCharArray();

        for(char c:carr)
        {
            if(lhm.get(c)!=null)
            {
                lhm.put(c,lhm.get(c)+1);
            }
            else
            {
                lhm.put(c,1);
            }
        }
        System.out.println(lhm);
    }
}
```

Q)Write a java program to check given string is balanced or not?
input: {[()]}
output: It is balanced string

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		String str="{[()]}";

		if(isBalanced(str))
			System.out.println("It is a balanced string");
		else
			System.out.println("It is not a balanced string");
	}
	//callie method
	public static  boolean isBalanced(String str)
	{
		Stack<Character> stack=new Stack<Character>();

		char[] carr=str.toCharArray();

		for(char c:carr)
		{
			if(c=='{' || c=='[' || c=='(')
				stack.push(c);
			else if(c=='}' || !stack.isEmpty() || stack.peek()=='{')
				stack.pop();
			else if(c==']' || !stack.isEmpty() || stack.peek()=='[')
				stack.pop();
			else if(c==')' || !stack.isEmpty() || stack.peek()=='(')
				stack.pop();
			else
				return false;
		}

		return stack.isEmpty();
	}
}
```

Q)Types of Data structure in java?

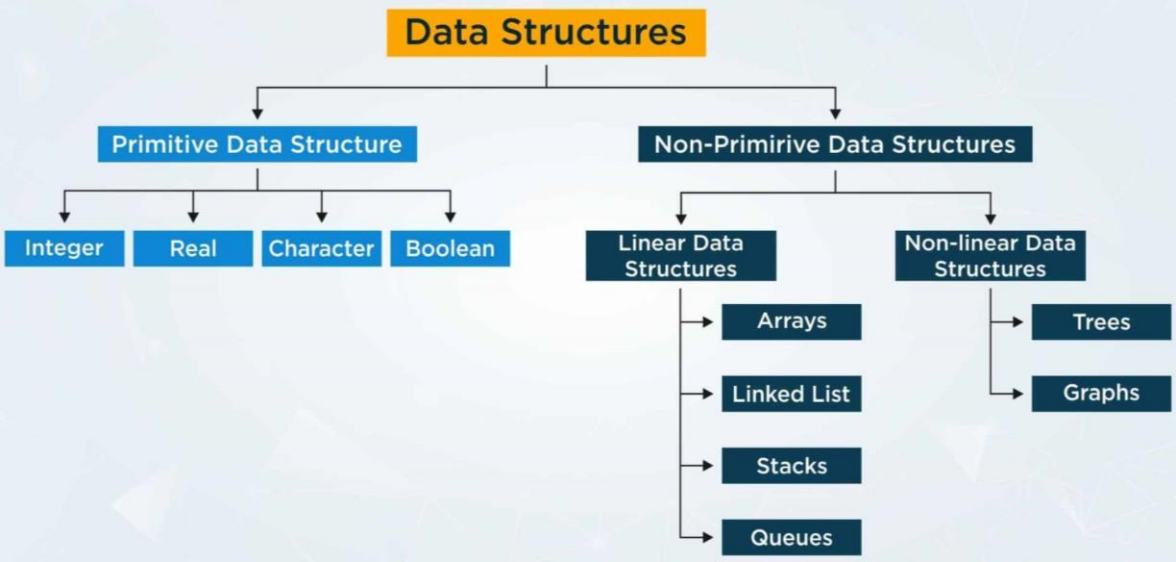


Diagram: java36.1

Q)Write a java program to display unique and duplicate elments from given array?
Input: 5 1 2 3 3 9 1 7 6 4 4 3
output: Unique : 5 1 2 3 9 7 6 4
Duplicates : 1 3 4

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,1,2,3,3,9,1,7,6,4,4,3};

		Set<Integer> duplicate=new LinkedHashSet<Integer>();
		Set<Integer> unique=new LinkedHashSet<Integer>();

		//for each loop
		for(int ele:arr)
		{
			if(!unique.add(ele))
			{
				duplicate.add(ele);
			}
			unique.add(ele);
		}

		System.out.println("Unique : "+unique);
		System.out.println("Duplicate :"+duplicate);
	}
}
```

Q)Write a java program to display unique elments from given array?
Input: 5 1 2 3 3 9 1 7 6 4 4 3
output: 5 1 2 3 9 7 6 4

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,1,2,3,3,9,1,7,6,4,4,3};

		Set<Integer> unique=new LinkedHashSet<Integer>();

		//for each loop
		for(int ele:arr)
		{
			unique.add(ele); // 5 1 2 3 9 7 6 4
		}

		//converting collection to array
		int[] resArr=new int[unique.size()];
		int j=0;
		for(Integer i:unique)
		{
			resArr[j++]=i;
		}

		//display
		for(int ele:resArr)
		{
			System.out.print(ele+" ");
		}
	}
}
```

Multithreading

Q)What is the difference between Thread and Process?

Thread

A thread is a leight weight sub-process.
We can run multiple threads concurently.
One thread can communicate with another thread.
ex: In java a class is one thread
In java a constructor is one thread
In java a block is one thread

Process

A process is a collection of threads.
We can run multiple process concurently.

One process can't communicate with another process because it is independent.
ex: Taking a java class by using zoom meeting is one process
Downloading a file from internet is one process
typing the notes in editor is one process

Multitasking
Executing several task simultenously such concept is called multitasking.
We have two types of multitasking.
1)Process based multitasking
Executing several task simultenously where each task is a independent process such types of multitasking is called process based multitasking.
2)Thread based multitasking
Executing several task simultenously where each task is a same part of a program such type of multitasking is called thread based multitasking.

Multithreading
Executing several threads simultenously such concept is called multithreading.
In multithreading only 10% of work should be done by a programmer and 90% of work will be done by a JAVA API.
The main important application area of multithreading are.
1) To implements multimedia graphics.
2) To develop video games
3) To develop animations.

Ways to create a thread in java
There are two ways to create or start or instantiate a thread in java.
1)By extending Thread class
2)By implementing Runnable interface

<u>1)By extending Thread class</u>

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantitate a thread
		MyThread t=new MyThread();
```

```
		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case1: Thread Schedular
If multiple threads are waiting for execution which thread will execute will decide by thread schedular.
What algorithm, behaviour and mechanism used by thread schedular is depends upon JVM vendor.
Hence we can't expect any execution order or exact output in multithreading.

case2: Difference between t.start() and t.run()
If we invoke t.start() method then a new object will created which is responsible to execute run() method automatically.

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantitate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
```

```
}
```

If we invoke t.run() method then no new thread will be created but run() method will execute just like normal method.

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantitate a thread
		MyThread t=new MyThread();

		//no new thread
		t.run();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case3: If we won't override run() method

If we won't override run() method then t.start() method will execute Thread class run() method automatically.

Thread class run() method is a empty implementation.

```
class MyThread extends Thread
{

}
class Test
{
	public static void main(String[] args)
	{
		//instantitate a thread
		MyThread t=new MyThread();
```

```
		//new thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case4: If we overload run() method
If we overload run() method then t.start() method always execute run() method with 0-argument method.

```
class MyThread extends Thread
{
	public void run()
	{
		System.out.println("0-arg method");
	}
	public void run(int i)
	{
		System.out.println("int-arg method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantitate a thread
		MyThread t=new MyThread();

		//new thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case5: Life cycle of a thread
Once if we create a thread object then our thread will be in new or born state.
Once if we call t.start() method then our thread goes to ready/runnable state.
If Thread schedular allocates to CPU then out thread enters to running state.
Once the run() method execution is completed then our thread goes to dead state.

```
MyThread t=new MyThread(); //instantiate a thread
t.start();
```

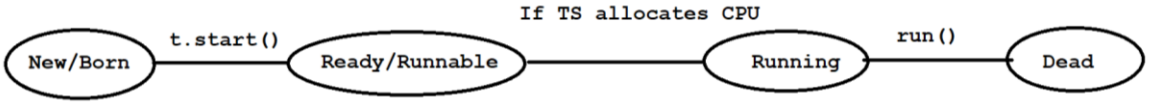


Diagram: java36.2

2)By implementing Runnable interface

```
class MyRunnable implements Runnable
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		 MyRunnable r=new  MyRunnable();

		Thread t=new Thread(r); // r is a targatable interface

		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

Setting and Getting name of a thread

In java , every thread has a name explicitly provided by the programmer and automatically generated by JVM.

We have following methods to set and get name of a thread.

ex: public final void setName(String name);
public final String getName();

```
class MyThread extends Thread
{
}
```

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(Thread.currentThread().getName());//main

		MyThread t=new MyThread();
		System.out.println(t.getName());//Thread-0

		Thread.currentThread().setName("Parent-Thread");
		System.out.println(Thread.currentThread().getName());//Parent-Thread

		t.setName("Child-Thread");
		System.out.println(t.getName());//Child-Thread
	}
}
```

Thread priority

In java, every thread has a priority explicitly provided by the programmer or automatically generated by JVM.

The valid range of thread priority is 1 to 10.Where 1 is a least priority and 10 is a highest priority.

Thread class defines following constants as a thread priority.

ex: Thread.MAX_PRIORITY - 10
Thread.MIN_PRIORITY - 1
Thread.NORM_PRIORITY - 5

We don't have such constants like LOW_PRIORITY and HIGH_PRIORITY.

A thread which is having highest priority will be executed first.

Thread schedular uses thread priority while allocating to CPU.

If multiple threads having same priority then we can't expect any execution order.

If we take more then 10 priority then we will get IllegalArgumentException.

We have following methods to set and get thread priority.

ex: public final void setPriority(int priority);
public final int getPriority();

```
class MyThread extends Thread
{
}
class Test
{
	public static void main(String[] args)
	{
		System.out.println(Thread.currentThread().getPriority());//5

		MyThread t=new MyThread();
		System.out.println(t.getPriority());//5
```

```
		Thread.currentThread().setPriority(9);
		System.out.println(Thread.currentThread().getPriority());//9

		t.setPriority(4);
		System.out.println(t.getPriority());//4
	}
}
```

Various Ways to prevent a thread in java

In three ways we can prevent(stop) a thread in java.

1)yield()

2)join()

3)sleep()

1)yield()

It will pause the current execution thread and gives the chance to other thread who is having same priority.

If there is no waiting thread or low priority thread then same thread will continue it's execution.

If multiple threads having same priority then we can't expect any execution order or exact output.

The thread which is yielded , when it will chance for execution is depends upon mercy of thread schedular.

ex: public static native void yield();

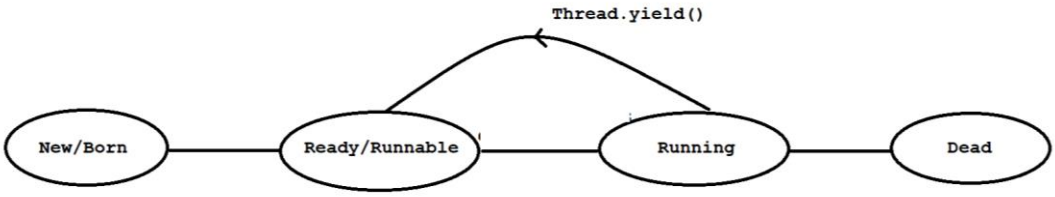


Diagram: java37.1

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.start();
		for(int i=1;i<=5;i++)
		{
```

```
                    Thread.currentThread().yield();
                    System.out.println("Parent-Thread");
                }
        }
}
```

2)join()

If a thread wants to wait untill the completion of some other threads then we need to use join() method.

A join() method will throw one checked exception so we must and should handle that exception by using try and catch block or by using throws statement.

ex: public final void join()throws InterruptedException
public final void join(long ms)throws InterruptedException
public final void join(long ms,int ns)throws InterruptedException

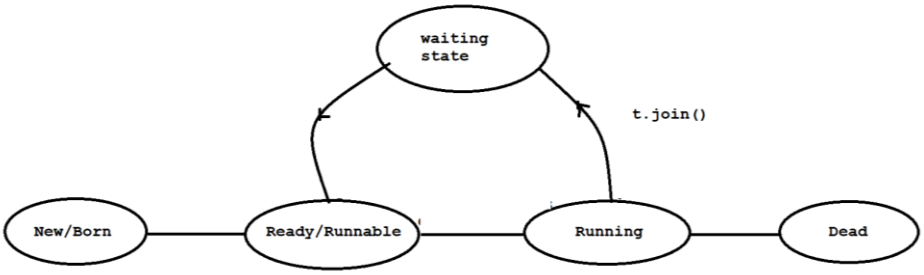


Diagram:java37.2

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Child-Thread");
                }
        }
}
class Test
{
        public static void main(String[] args)throws  InterruptedException
        {
                MyThread t=new MyThread();
                t.start();
                t.join();
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Parent-Thread");
                }
        }
}
```

3)sleep()

If a thread don't want to perform any operation on perticular amount of time then we need to use sleep() method.

A sleep() will throw one checked exception so we must and should handle that exception by using try and catch block or by using throws statement.

ex: public static native void sleep()throws InterruptedException
public static native void sleep(long ms)throws InterruptedException
public static native void sleep(long ms,int ns)throws InterruptedException

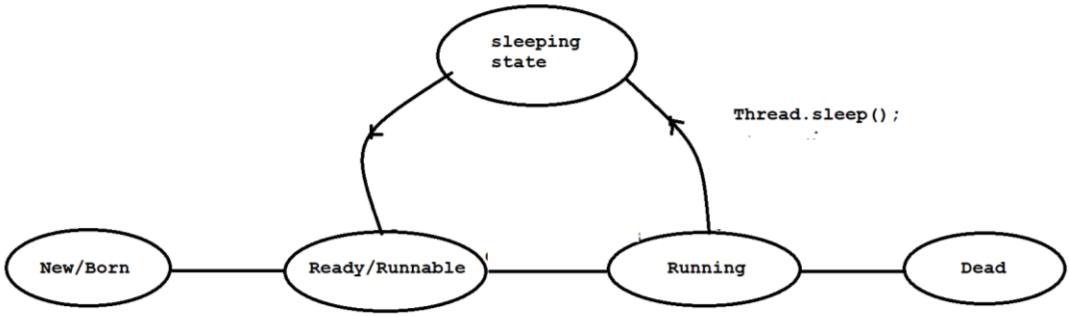


Diagram:java37.3

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
			try
			{
				Thread.sleep(2000);
			}
			catch (InterruptedException ie)
			{
				ie.printStackTrace();
			}
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.start();
		for(int i=1;i<=5;i++)
		{  System.out.println("Parent-Thread");  }
	}
}
```

Daemon Thread

It is a service provider thread which provides services to user threads.

The life of daemon thread is completely depends upon user threads.When user thread died then daemon thread will terminate automatically.

There are many daemon thread are running internally like Garbage Collector, Finalizer and etc.

To start a daemon thread we need to use setDaemon(true) method.

To check either thread is a daemon or not we need to use isDaemon() method.

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println(Thread.currentThread().isDaemon());
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.setDaemon(true);
		t.start();
		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

Problem without synchronization

If there is a no synchronization then we will face following problems.

1) Data inconsistency.

2) Thread interference.

```
class Table
{
	void printTable(int n)
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println(n*i);
			try
			{
				Thread.sleep(2000);
```

```
                        }
                        catch (InterruptedException ie)
                        {
                                ie.printStackTrace();
                        }
                }
        }
}
class MyThread1 extends Thread
{
        Table t;

        MyThread1(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(5);
        }
}
class MyThread2 extends Thread
{
        Table t;

        MyThread2(Table t)
        {
                this.t=t;
        }

        public void run()
        {
                t.printTable(10);
        }
}
class Test
{
        public static void main(String[] args)
        {
                Table obj=new Table();
                MyThread1 t1=new MyThread1(obj);
                MyThread2 t2=new MyThread2(obj);
                t1.start();
                t2.start();
        }
}
```

synchronization

A synchronized keyword is applicable for methods and blocks.

A synchronization is allowed one thread to execute given object.Hence we achieve thread safety.

The main advantage of synchronization is we solve data inconsistence problem.

The main disadvantage of synchronization is ,it will increase waiting time of a thread which reduce the performance of the system.

If there is no specific requirement then it is never recommanded to use synchronization concept.

synchronization internally uses lock mechanism.

Whenever a thread wants to access object , first it has to acquire lock of an object and thread will release the lock when it completes it's task.

When a thread wants to execute synchronized method.It automatically gets the lock of an object.

When one thread is executing synchronized method then other threads are not allowed to execute other synchronized methods in a same object concurently.But other threads are allowed to execute non-synchronized method concurently.

```
class Table
{
	synchronized void printTable(int n)
	{
		for(int i=1;i<=5;i++)
		{
				System.out.println(n*i);
				try
				{
					Thread.sleep(2000);
				}
				catch (InterruptedException ie)
				{
					ie.printStackTrace();
				}
		}
	}
}
class MyThread1 extends Thread
{
	Table t;
	MyThread1(Table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printTable(5);
	}
}

class MyThread2 extends Thread
```

```
{
	Table t;
	MyThread2(Table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printTable(10);
	}
}

class Test
{
	public static void main(String[] args)
	{
		Table obj=new Table();
		MyThread1 t1=new MyThread1(obj);
		MyThread2 t2=new MyThread2(obj);

		t1.start();
		t2.start();
	}
}
```

synchronized block

If we want to perform synchronization on specific resource of a program then we need to use synchronization.

ex: If we have 100 lines of code and if we want to perform synchronization only for 10 lines then we need to use synchronized block.

If we keep all the logic in synchronized block then it will act as a synchronized method.

```
class Table
{
	 void printTable(int n)
	{
		synchronized(this)
		{
		for(int i=1;i<=5;i++)
		{
				System.out.println(n*i);
				try
				{
					Thread.sleep(2000);
				}
				catch (InterruptedException ie)
				{
```

```
                                        ie.printStackTrace();
                                    }
                }
                }//sync
        }
}
class MyThread1 extends Thread
{
        Table t;
        MyThread1(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(5);
        }
}

class MyThread2 extends Thread
{
        Table t;
        MyThread2(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(10);
        }
}

class Test
{
        public static void main(String[] args)
        {
                Table obj=new Table();
                MyThread1 t1=new MyThread1(obj);
                MyThread2 t2=new MyThread2(obj);

                t1.start();
                t2.start();
        }
}
```

3)Static synchronization

In static synchronization the lock will be on class but not on object.

If we declare any static method as synchronized then it is called static synchronization method.

```
class Table
{
	static synchronized void printTable(int n)
	{

		for(int i=1;i<=5;i++)
		{
				System.out.println(n*i);
				try
				{
					Thread.sleep(2000);
				}
				catch (InterruptedException ie)
				{
					ie.printStackTrace();
				}
		}

	}
}
class MyThread1 extends Thread
{

	public void run()
	{
		Table.printTable(5);
	}
}

class MyThread2 extends Thread
{

	public void run()
	{
		Table.printTable(10);
	}
}

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

```
                MyThread1 t1=new MyThread1();
                MyThread2 t2=new MyThread2();

                t1.start();
                t2.start();
        }
}
```

.Inter-Thread Communication

Two threads can communicate with one another by using wait(),notify() and notifyAll() method.
The Thread which is expecting updations it has to wait() method and the thread which is performing updations it has to call notify() method.
wait(),notify() and notifyAll() method present in Object class but not in Thread class.
To call wait(),notify() and notifyAll() method our current thread must be in a synchronized area otherwise we will get IllegalMonitorStateException.
Once a thread calls wait() method on a given object ,1st it will release the lock of that object immediately and entered into waiting state.
Once a thread calls notify() and notifyAll() method on a given object.It will release the lock of that object but not immediately.
Except wait(),notify() and notifyAll() method ,there is no such concept where lock release can happen.

```
class MyThread extends Thread
{
        int total=0;
        public void run()
        {

                synchronized(this)
                {
                                System.out.println("Child Thread started calculation");
                                for(int i=1;i<=10;i++)
                                {
                                        total=total+i;
                                }
                                System.out.println("Child thread giving notification");
                                this.notify();
                }
        }
}
class Test
{
        public static void main(String[] args)throws InterruptedException
        {
                MyThread t=new MyThread();
                t.start();
                synchronized(t)
```

```
                {
                        System.out.println("Main Thread waiting for updating");
                        t.wait();
                        System.out.println("Main -Thread got notification ");
                        System.out.println(t.total);
                }
        }
}
```

DeadLock in java

DeadLock will occur in a suitation when one thread is waiting to access object lock which is acquired by another thread and that thread is waiting to access object lock which is acquired by first thread.

Here both the threads are waiting release the thread but no body will release such situation is called DeadLock.

```
class Test
{
        public static void main(String[] args)
        {
                final String res1="hi";
                final String res2="bye";

                Thread t1=new Thread()
                {
                        public void run()
                        {
                                synchronized(res1)
                                {
                                        System.out.println("Thread1: Locking Resource 1");
                                        synchronized(res2)
                                        {
                                                System.out.println("Thread1: Locking Resource2");
                                        }
                                }
                        }
                };

                Thread t2=new Thread()
                {
                        public void run()
                        {
                                synchronized(res2)
                                {
                                        System.out.println("Thread2: Locking Resource 2");
                                        synchronized(res1)
                                        {
```

```
                                        System.out.println("Thread1: Locking Resource 1");
                                        }
                                }
                        }
                };
                t1.start();
                t2.start();
        }
}
```

Drawbacks of multithreading

1)DeadLock
2)Thread Starvation

Java 8 Features

1) java.time package
2) Functional interface
3) Lamda Expression
4) default methods in interface
5) static methods in interface
6) Stream API
7) forEach() method
and etc.

Functional interface

Interface which contains only one abstract method is called functional interface.
It can have any number of default methods and static methods.

ex: Runnable ---- run()
Comparable ---- compareTo()
ActionListener --- actionPerformed()
and etc.

Functional interface is also known as SAM or Single Abstract Method interface.
Functional interface is used to achieve functional programming.

ex: i=f1(){}
f1(f2(){})
{}

@FunctionalInterface annotation is used to declare function interface and it is optional.

syntax:
```
@FunctionalInterface
interface  <interface_name>
{
        - // 1 abstract method
        - // default methods
        - // static methods
}
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract void m1();
}
class B implements A
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
	}
}
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract void m1();
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A()
		{
			public void m1()
			{
				System.out.println("From M1-Method");
			}
		};
		a.m1();
	}
}
```

Lamda Expression

Lamda expression introduced in java 8.

Lamda expression is used to concise the code.

Lamda expression we can use when we have functional interface.
Lamda expression consider as method not a class.
The main objective of lamda expression is to achieve functional programming.
Lamda expression does not allow name, returntype and modifier.

ex: java method

```
public void m1()
{
        System.out.println("Hello World");
}
```

lamda expression

```
()->
{
        System.out.println("Hello World");
};
```

ex:

```
@FunctionalInterface
interface A
{
        public abstract void m1();
}
class Test
{
        public static void main(String[] args)
        {
                A a=()->
                        {
                                System.out.println("M1-Method");
                        };
                a.m1();
        }
}
```

ex:

```
@FunctionalInterface
interface A
{
        public abstract void m1(int i,int j);
}
class Test
{
        public static void main(String[] args)
        {
                A a=(int i,int j)->
                        {
```

```
                        System.out.println(i+j);
                    };
                a.m1(10,20);
        }
}
```

ex:

```
@FunctionalInterface
interface A
{
        public abstract String m1();
}
class Test
{
        public static void main(String[] args)
        {
                A a=()->
                    {
                        return "Hello World";
                    };
                System.out.println(a.m1());
        }
}
```

default methods in interface

A default method introduced in java 8.
To declare default methods in interface we will use "default" keyword.
A default method is a non-abstract method.
A default method can be override.
ex:

```
@FunctionalInterface
interface A
{
        //abstract method
        public abstract void m1();

        //default method
        default void m2()
        {
                System.out.println("M2-Method");
        }
}
class B implements A
{
        public void m1()
        {
```

```
		System.out.println("M1-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
		a.m2();
	}
}
```

ex:

```
@FunctionalInterface
interface A
{
	//abstract method
	public abstract void m1();

	//default method
	default void m2()
	{
		System.out.println("M2-Method");
	}
}
class B implements A
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
	public void m2()
	{
		System.out.println("Override-M2-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
		a.m2();
	}
}
```

With the help of default methods we can achieve multiple inheritance.
ex:

```
interface Right
{
	default void m1()
	{
		System.out.println("Right-M1-Method");
	}
}
interface Left
{
	default void m1()
	{
		System.out.println("Left-M1-Method");
	}
}
class Middle implements Right,Left
{
	public void m1()
	{
		System.out.println("Middle-M1-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Middle m=new Middle();
		m.m1();
	}
}
```

o/p: Middle-M1-Method

ex:

```
interface Right
{
	default void m1()
	{
		System.out.println("Right-M1-Method");
	}
}
interface Left
{
	default void m1()
	{
		System.out.println("Left-M1-Method");
```

```
	}
}
class Middle implements Right,Left
{
	public void m1()
	{
		Right.super.m1();
	}
}
class Test
{
	public static void main(String[] args)
	{
		Middle m=new Middle();
		m.m1();
	}
}
```
o/p: Right-M1-Method

ex:

```
interface Right
{
	default void m1()
	{
		System.out.println("Right-M1-Method");
	}
}
interface Left
{
	default void m1()
	{
		System.out.println("Left-M1-Method");
	}
}
class Middle implements Right,Left
{
	public void m1()
	{
		Left.super.m1();
	}
}
class Test
{
	public static void main(String[] args)
	{
		Middle m=new Middle();
```

```
                m.m1();
        }
}
```
o/p: Left-M1-Method

static methods in interface

A static method introduced in java 8.
To declare static methods in interface we will use "static" keyword.
A static method is a non-abstract method.
A static method can't be override.
ex:

```
interface A
{
        //static method
        static void m1()
        {
                System.out.println("static-method");
        }
}
class Test
{
        public static void main(String[] args)
        {
                A.m1();
        }
}
```

Stream API

Stream API introduced in java 8.
A Stream is an interface which is present in java.util.stream package.
It is used to perform bulk operations on Collections.

```
import java.util.*;
import java.util.stream.*;
class Test
{
        public static void main(String[] args)
        {
                List<Integer> list=Arrays.asList(2,5,7,4,6,9,1);

                List<Integer> even=list.stream().filter(i->i%2==0).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(2,5,7,4,6,9,1);

		List<Integer> odd=list.stream().filter(i->i%2!=0).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(2,5,7,4,6,9,1);

		List<Integer> newList=list.stream().sorted().collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(2,5,7,4,6,9,1);

		List<Integer>
newList=list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(2,5,7,4,6,9,1);

		long max=list.stream().max((i1,i2)->i1.compareTo(i2)).get();

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(2,5,7,4,6,9,1);

		long min=list.stream().min((i1,i2)->i1.compareTo(i2)).get();

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> marks=Arrays.asList(24,58,72,49,16,90,81);

		List<Integer> list=marks.stream().map(i->i+10).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> marks=Arrays.asList(24,58,72,49,16,90,81);

		long failed=marks.stream().filter(i->i<35).count();

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

forEach() method

The forEach() method in Java is a utility method to iterate over a Collection (list, set or map) or Stream.

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(24,58,72,49,16,90,81);

		list.forEach(System.out::println);
	}
}
```

Interview Question

Q)Write a java program to sort employees by id using java 8 stream api?

```
import java.util.*;
import java.util.stream.*;

class Employee
{
	private int empId;
	private String empName;
	private double empSal;

	//default constructor
	Employee()
	{

	}
```

```
	//parameterized constructor
	Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}

	//setter methods
	public void setEmpId(int empId)
	{
		this.empId=empId;
	}
	public void setEmpName(String empName)
	{
		this.empName=empName;
	}
	public void setEmpSal(double empSal)
	{
		this.empSal=empSal;
	}
	//getter methods
	public int getEmpId()
	{
		return empId;
	}
	public String getEmpName()
	{
		return empName;
	}
	public double getEmpSal()
	{
		return empSal;
	}
}

class Test
{
	public static void main(String[] args)
	{
		List<Employee> employees=new ArrayList<Employee>();
		employees.add(new Employee(104,"Alan",4000d));
		employees.add(new Employee(101,"Jose",1000d));
		employees.add(new Employee(103,"Kelvin",3000d));
		employees.add(new Employee(102,"Nelson",2000d));
```

```
            List<Employee>
list=employees.stream().sorted(Comparator.comparingInt(Employee::getEmpId)).collect(Collect
ors.toList());

            list.forEach(employee -> System.out.println(employee.getEmpId() +"
"+employee.getEmpName() +" "+employee.getEmpSal()));

        }
}
```

SDLC

SDLC stands for Software Development Life Cycle.
It is a process adopt by IT industry to develop accurate and quality of softwares.
There are six phases are there in SDLC.

1. Feasibility study
2. Analysis
3. Designing
4. Coding
5. Testing
6. Delivery and Maintainence

1)Feasibility study

Feasibility study completed depends upon TELOS formulea.

ex: T - Technical Feasibility
E - Ecnomical Feasibility
L - Legal Feasibility
O - Operational Feasibility
S - Scheduled Feasibility

All the above information they will keep in a document called BDD.
BDD stands for Business Designed Document.

2)Analysis

In this phase , system analyst or product owner will involved.
They will seperate system requirements and software requirements.
They will keep this information in a document called SRS.
SRS stands Software Requirement Specification.

3)Designing

Designing can be performed in two ways.

<u>i)High level designing</u>

Manager is responsible to perform high level designing.
In high level designing ,we will design main modules.

<u>ii)Low level designing</u>

Team Lead/Project Lead is resposible to perform low level designing.
In low level designing , we will design sub-module/child modules.

All this above information we will keep in a document called PDD/TDD.
Here PDD stands for Project Design Document.
Here TDD stands for Technical Design Document.

4)Coding
In coding phase, developers will involved.
Developers are responsible to generate the build(source code).
Once our build is ready then developer even responsible to perform white box testing.

5)Testing Phase
In this phase, testing team or QA team will involved.
The will test the code by using one software component called STLC.
STLC stands for Software Testing Life Cycle.
Testing Team or QA team is responsible to perform black box testing.

6)Delivery & maintainence phase
Before delivery we will perform UAT testing.
UAT stands for User Acceptance Testing.
UAT testing is classified into two types.
- i)Alpha testing
- ii)Beta testing

Project
Project is a collection of modules.
ex: customer module
login module
registration module
payment module
report generation module
and etc.
Every project contains two domains.
1. Technical Domain
Using which technology we developed our project.
ex: Java
2. Functional Domain
It describes state of a project.
ex: Healthcare domain
Insaurance domain
Banking domain
ERP domain
and etc.