Core java notes by madhu…

Java Introduction :

Author	James gosling
Vendor	Sun micro system
Project name	Green project
Type	Open source and free software
Intial name	Oak language
Present name	Java
Intial version	Jdk 1.0
Latest version	Jdk 1.8
Objective	To develop any kind of applications
Slogan	Wora(write once run any where)

Example java program :

```
package com.madhu.basics;

public class Sample {

	public static void main(String[] args) {

		System.out.println("Techfort software servcies pvt ltd");

	} }
```

Compilation : javac Sample.java

Execution : java Sample

Output : Techfort software services pvt ltd

Importance of core java :

1) Java is used to develop Desktop Applications such as MediaPlayer,Antivirus etc.

2) Java is Used to Develop Web Applications such as gmail.com, irctc.co.in etc.

3) Java is Used to Develop Enterprise Application such as Banking applications.

4) Java is Used to Develop Mobile Applications.

5) Java is Used to Develop Embedded System.

6) Java is Used to Develop SmartCards.

7) Java is Used to Develop Robotics.

8) Java is used to Develop Games etc.

Java Versions :

Java Versions	Year
Jdk Alpha & Beta	1995
Jdk 1.0	1996
Jdk 1.1	1997
J2se 1.2	1998
J2se 1.3	2000
J2se 1.4	2002
J2se 1.5	2004
Jse 6	2006
Jse 7	2011
Jse 8	

Parts of java language :

Generally there is no core java and advanced java in java.

As per the sun micro system standard the java language is divided into three types.

1) J2SE/JSE(java 2 standard edition)

2) J2EE/JEE(java 2 enterprise edition)

3) J2ME/JME(java 2 micro edition)

J2SE:-

By using j2se we are able to develop the standalone applications.

Ex:- notepad, WordPad, paint, Google Talk……..etc

Standalone applications:-

1) Standalone applications are the java applications which don’t need the client server architecture.

2) The standalone applications applicable for the only one desktop hence it is called

desktop applications or window based applications.

J2EE:-

By using j2ee we are able to develop the web based applications.

Ex:- Gmail, yahoo mail, bank, reservation……etc

Web-applications:-

1) Web applications are the java applications which needs client and server concept.

2) Web applications must need the internet connections to access the application.

J2ME:-

By using j2me we are able to develop the applications that applications only run on mobile devices.

Differences between c, cpp and java :

S no	C language	Cpp language	Java language
1	The program execution starts from main method and main method is called by Operating system.	The program execution starts from main method called by operating system.	The program execution starts from main method called by JVM (java virtual machile).
2	In the c-language the predefined support is maintained in the form of header files Ex:- stdio.h,conio.h	In the cpp language the predefined is maintained in the form of header files. Ex:- iostream.h	In the java language the predefined is maintained in the form of packages. Ex:- java.lang, java.io, java.net, java.awt
3	the header files contains predefined functions. Ex:- printf,scanf	The header files contains predefined functions. Ex:- cout,cin	The packages contains predefined classes. Ex:- String,System
4	To make available predefined support into our program we have to use #include statement. Ex:- #include<stdio.h>	To make available predefined support into our program we have to use #include statement. Ex:- #include<iostream>	To make available predefined support into our program we have to use import statement. Ex:- import java.lang.*;

Java Features :

1. Simple:-

Java is a simple programming language because:

Java technology has eliminated all the difficult and confusion oriented concepts like pointers, multiple inheritance in the java language.

The c,cpp syntaxes easy to understand and easy to write. Java maintains C and CPP syntax mainly hence java is simple language.

Java tech takes less time to compile and execute the program.

2. Object Oriented:-

Java is object oriented technology because to represent total data in the form of object.

By using object reference we are calling all the methods, variables which is present in that class.

The total java language is dependent on object only hence we can say java is a object oriented technology.

3. Platform Independent :-

Compile the Java program on one OS (operating system) that compiled file can execute in any OS(operating system) is called Platform Independent Nature. The java is platform independent language. The java applications allows its applications compilation one operating system that compiled (.class) files can be executed in any operating system.

4. Architectural Neutral:-

Java tech applications compiled in one Architecture (hardware----RAM, Hard Disk) and that Compiled program runs on any hardware architecture(hardware) is called Architectural Neutral.

5. Portable:-

In Java tech the applications are compiled and executed in any OS(operating system) and any Architecture(hardware) hence we can say java is a portable language.

6. Robust:-

Any technology if it is good at two main areas it is said to be ROBUST

1 Exception Handling

2 Memory Allocation

JAVA is Robust because

a. JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.

b. JAVA is having very good memory management system that is Dynamic Memory (at runtime the memory is allocated) Allocation which allocates and deallocates memory for objects at runtime.

7. Secure:-

To provide implicit security Java provide one component inside JVM called Security Manager.

To provide explicit security for the Java applications we are having very good predefined library in the form of java.Security.package.

Web security for web applications we are having JAAS(Java Authentication and Authorization Services) for distributed applications.

8. Dynamic:-

Java is dynamic technology it follows dynamic memory allocation(at runtime the memory is allocated) and dynamic loading to perform the operations.

9. Distributed:-

By using JAVA technology we are preparing standalone applications and Distributed applications.

Standalone applications are java applications it doesn’t need client server architecture. web applications are java applications it need client server architecture.

Distributed applications are the applications the project code is distributed in multiple number of jvm’s.

10. Multithreaded: -

Thread is a light weight process and a small task in large program.

If any tech allows executing single thread at a time such type of technologies is called single threaded technology.

If any technology allows creating and executing more than one thread called as Multithreaded technology called JAVA.

11. Interpretive:-

JAVA tech is both Interpretive and Completive by using Interpretator we are converting source code into byte code and the interpretor is a part of JVM.

12. High Performance:-

If any technology having features like Robust, Security, Platform Independent, Dynamic and so on then that technology is high performance.

Install the software and set the path :-

Download the software from internet based on your operating system. The software is different from 32-bit operating and 64-bit operating system.

To download the software open the fallowing web site.

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

for 32-bit operating system please click on Windows x86 :- 32- bit operating system

for 64-bit operating system please click on Windows x64 :- 64-bit operating system

After installing the software the java folder is available in the following location

Local Disk c: program Files--java--jdk(java development kit), jre(java runtime environment)

To check whether the java is installed in your system or not go to the command prompt. To open the command prompt

Start -- run-- open: cmd -- ok

Command prompt is opened.

In the command prompt type :- javac

‘javac’ is not recognized is an internal or external command, operable program or batch file.

Whenever we are getting above information at that moment the java is installed but the java is not working properly.

C:/>javac

Whenever we are typing javac command on the command prompt

1) Operating system will pickup javac command search it in the internal operating system calls. The javac not available in the internal command list .

2) Then operating system goes to environmental variables and check is there any path is sets or not. up to now we are not setting any path. So operating system don’t know anything about javac command Because of this reason we are getting error message.

Hence we have to environmental variables. The main aim of the setting environmental variable is to make available the fallowing commands javac,java,javap (softwares) to the operating system.

To set the environmental variable:-

My Computer (right click on that) – properties – Advanced -- Environment Variables---

User variables – new -- variable name : Path

Variable value : C:\programfiles\java\jdk1.6.0_11\bin;.; --ok -- ok

Now the java is working good in your system. open the command prompt to check once

C:>javac ---- now list of commands will be displayed

Class Contains Five elements:-

Class Test

{

1. variables

2. methods

3. constructors

4. instance blocks

5. static blocks

}

Main Method:- Public static void main(String[] args)

Public :- To provide access permission to the jvm declare main is publc.

Static :- To provide direct access permission to the jvm declare main is static(with out creation of object able to access main method)

Void :- don’t return any values to the JVM.

String[] args :- used to take command line arguments(the arguments passed from command prompt)

String represents possible to take any type of argument.

[ ] represent possible to take any number of arguments.

Tokens :- Smallest individual part in a java program is called Token. It is possible to provide any number of spaces in between two tokens.

Ex:-

Class Test

{

Public static void main(String[] args)

{ int a=10;

System.out.println(“java tokens”);

}

}

Tokens are -- class,test,{,”,[--------etc

Print() vs Println ():-

Print():-

Print is used to print the statement into the console and the control is available in the same line.

Ex:-

```
System.out.print(“Techfort”);

System.out.print(“softwate services pvt ltd”);
```

Output:- Techfort software services pvt ltd

Println():-

In the println statement Print is used to print the statement into the console and ln represent go to the new line now the control is available in the next line.

Ex:-

```
System.out.println(“Techfort”);

System.out.println(“software services pvt ltd”);
```

Output:- Techfort

Software services pvt ltd

Identifiers:-

Any name in the java program like variable name,class name, method name, interface name is called identifier.

Ex:-

```
class Test

{

void add()

{

int a=12;

int b=13;

}

}
```

Test, add, a, b are identifiers

Java Naming Conventions :-

Java is a case sensitive language so the way of writing code is important.

1. All Java classes, Abstract classes and Interface names should start with uppercase letter ,if any class contain more than one word every innerword also start with capital letters.

Ex:- String

StringBuffer

2. All java methods should start with lower case letters and if the method contains more than one word every innerword should start with capital letters.

Ex :- post()

toString()

toUpperCase()

3. All java variables should start with lowercase letter and inner words start with uppercase letter.

Ex:- pageContent

bodyContent

4. All java constant variables should be in uppercase letter.

Ex:- MIN_PRIORITY

MAX_PRIORITY

NORM_PRIORITY

5. All java packages should start with lower case letters only.

Ex:- java.awt

Java.io

Java Comments :-

To provide the description about the program we have to use java comments.

There are 3 types of comments present in the java language.

1) Single line Comments:-

By using single line comments we are providing description about our program within a single line.

Starts with // (double slash)

Syntax:- //description

2) Multi line Comments:-

This comment is used to provide description about our program in more than one line.

Syntax:- /*…….line-1

……..line-2

*/

3) Documentation Comments:-

This comment is used to provide description about our program in more than one page.

In general we are using document comment to prepare API kind of documents but it is not suggestable.

Syntax:- /*…….line-1

*……..line-2

line-3

*/

Control statements :-

There are 2 types of statements in java :

Conditional statements

Iteration statements

Conditional statements : if, if-else, switch

If syntax:-

if (condition)

{

Statements;

}

The curly brasses are optional whenever we are taking single statements.

The curly brasses are mandatory whenever we are taking multiple statement.

Ex:-

class Test

```
{
public static void main(String[] args)
{
int a=10;
if (a>5)
{
System.out.println("if body / true body");
}
}
}
```

If-else syntax:-

```
if (condition)
{
if body; (true body)
}
else
{
else body; (false body)
}
```

The curly brasses are optional whenever we are taking single statements.

The curly brasses are mandatory whenever we are taking multiple statements.

Ex:-

```
class Test
{
public static void main(String[] args)
{
int a=10;
int b=20;
```

```
if (a<b)

{

System.out.println("if body / true body");

}

else

{

System.out.println("else body/false body ");

}

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

}

}
```

Switch statement:-

1) Switch statement is used to take multiple selections.

2) Curly brasses are mandatory if we are not taking we are getting compilation error.

3) Inside the switch It is possible to declare any number of cases but is possible to declare only one default.

4) Switch is taking the argument the allowed arguments are

a. Byte

b. Short

c. Int

d. Char

e. String (allowed in 1.7 version)

5) Float and double and long is not allowed for a switch argument because these are having more number of possibilities (float and double is having infinity number of possibilities) hence inside the switch statement it is not possible to provide float and double and long as a argument.

6) If the case is matched particular case will be executed if there is no case is matched default case is executed.

Iteration Statements:-

If we want to execute group of statements repeatedly or more number of times then we should go for iteration statements.

Three types of iteration statements present in the java language :

1) for

2) while

3) do-while

for syntax:-

```
for (part 1;part 2 ;part 3 )

{

Body;

}
```

Ex:-

```
for (initialization ;condition ;increment/decrement )

{

Body;

}
```

1) The for loop contains three parts initialization, condition, increment/decrement part.

2) Each and every part is separated by semicolon and it is mandatory.

The curly brasses are optional whenever we are taking single statement.

The curly brasses are mandatory whenever we are taking more than one statements.

Ex:-

```
class Test

{

Public static void main(String args[])

{

for(int i=0;i<5;i++)

{
```

```
System.out.println(“Techfort software services pvt ltd”);

}

}

}
```

Do-While:-

1) If we want to execute the loop body at least one time them we should go for do-while statement.

2) In the do-while first body will be executed then only condition will be checked.

3) In the do-while the while must be ends with semicolon otherwise we are getting compilation error.

4) do is taking the body and while is taking the condition and the condition must be Boolean condition.

Syntax:-

```
do

{

//body of loop

} while(condition);
```

Ex :-

```
class Test

{

public static void main(String[] args)

{

int i=0;

do

{

System.out.println("Kodanda Ramu");

i++;

}while (i<10);

}

}
```

Transfer statements:- by using transfer statements we are able to transfer the flow of execution from one position to another position .

1. break

2. continue

3. return

4. try

break:- we are able to use the break statement only two places if we are using any other place the compiler will raise compilation error.

a. Inside the switch statement.

b. Inside the loops.

Ex :- break means stop the execution come out of loop.

```
class Test

{

public static void main(String[] args)

{

for (int i=0;i<10;i++)

{

if (i==5)

{

break;

}

System.out.println(i);

}

}

}
```

Continue:- (skip the current iteration continue the rest of the iterations normally)

```
class Test

{
```

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

{

for (int i=0;i<10;i++)

{

if (i==5)

{

continue;

}

System.out.println(i);

}

}

}
```

Data Types:-

1) Data types are used to represent the type of the variable and type of the expression.

2) Data types are used to specify the how much memory is allocated for variables.

Data type	Size	Range
Byte	1	-128 to 127
Short	2	-32768 to 32767
Int	4	-2147483648 to 2147483647
Long	8	-2`31 to 2`31-1
Float	4	-3.4e38 to 3.4e308
Double	8	-1.7e308 to 1.7e308
Char	2	0 to 65535
boolean	N A	Not Applicable

Variables :- used to store the values. While declaring variable we must specify the type of the variable by using data types concept.

In java, we have 3 kinds of variables.

Local variables

Instance variables

Static variables

Local variables : The variables which are declare inside a method & inside a block & inside a constructor is called local variables

The scope of local variables are inside a method or inside a constructor or inside a block.

Ex:-

```
class Test

{

public static void main(String[] args)

{

int a=10; // Local variables

int b=20;

System.out.println(a+b);

}

}
```

Instance variables:-

The variables which are declare inside a class and outside of the methods is called instance variables.

We are able to access instance variables only inside the class any number of methods.

Ex:-

```
class Test

{

int a=10;

int b=20;

void add()

{

System.out.println(a+b);

}

public static void main(String[] args)

{
```

```
Test t=new Test();

System.out.println(t.a+t.b);

t.add();

}

}
```

3. Static variables:-

The instance variables which are declared as a static modifier such type of variables are called static variables.

We are able to access static variables within the class any number of methods.

```
class Test

{

static int a=10;

static int b=20;

public static void main(String[] args)

{

System.out.println(a+b);

}

void add()

{

System.out.printl(a+b);

}

}
```

Calling of static variables:-

a. Directly possible.

b. By using class name possible.

c. By using reference variable possible.

Ex:-

```
class Test
```

```
{
static int x=100;
public static void main(String[] args)
{
//1-way(directly possible)
System.out.println(a);
//2-way(By using class name)
System.out.println(Test.a);
//3-way(By using reference variable)
Test t=new Test();
System.out.println(t.a);
}
}
```

Instance vs Static variables:-

1. Instance variable for the each and every object one separate copy is maintained.

2. Static variable for all objects same copy is maintained. One Object change the value another object is affected.

Ex:-

```
class Test
{
int a=10;
static int b=20;
public static void main(String args[])
{
Test t1=new Test();
System.out.println(t1.a);//10
System.out.println(t1.b);//20
t1.a=444;
```

```
t1.b=555;

Test t2=new Test();

System.out.println(t2.a);//10

System.out.println(t2.b);//555

t2.b=111;

System.out.println(t2.b);//111

Test t3=new Test();

System.out.println(t3.a);//10

System.out.println(t3.b);//111

Test t4=new Test();

System.out.println(t4.a);//10

System.out.println(t4.b);//111

}

}
```

Class and Object :

Class is a group of objects that have common property.

Java is classes based based language we are able to design the program by using classes and objects.

Object is a realworld entity. Object orientation is methodology to design a program by using classes and objects.

Object is physical entity where as class is a logical entity.

A class is a template or blue print from which type of objects are created.

Object is nothing but instance of a class.

Every objects contains 3 characterstics :

1. State(represent data of an object)

2. Behavior(represent behavior of an object)

3. Identity(used to identify the objects uniquely).

Example program for creating class and object :

```
Class Test

{

Public static void main(String args[])

{

System.out.println(“Techfort software services pvt ltd”);

}

}
```

Creation of object :

```
Test t=new Test();
```

Test	Class
t	Reference variable
new	Keyword/operator
Test()	Constructor used to intialize

Arrays :

Arrays are used to store the multiple numbers of elements of single type.

The length of the array is established at the time of array creation. After creation the length is fixed.

The items presented in the array are classed elements. Those elements can be accessed by index values. The index is begins from (0).

Advantages of array:-

1) Length of the code will be decreased

2) We can access the element present in the any location.

3) Readability of the code will be increased.

single dimensional array declaration:-

```
int[] a;

int []a;

int a[];
```

declaration & instantiation & initialization :-

```
approach 1:- int a[]={10,20,30,40};
```

approach 2:- int[] a=new int[100];

a[0]=10;

a[1]=20;

a[2]=30;

a[4]=40;

Example 1 :

```
package com.madhu.basics;

public class ArrayDemo1 {

	public static void main(String[] args) {

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

		System.out.println(a[0]);

		System.out.println(a[1]);

		System.out.println(a[2]);

		System.out.println(a[3]);

	}

}
```

Example 2 : (using for loop)

```
package com.madhu.basics;

public class ArrayDemo2 {

public static void main(String[] args) {

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

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

	{

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

	}

}

}
```

Example 3 : (using for-each loop) (1.5 version)

```
package com.madhu.basics;
public class ArrayDemo3 {
public static void main(String[] args) {
       int a[]={10,20,30,40};
       for(int a1:a)
       {
              System.out.println(a1);
       }
}
}
```

declaration of two dimensional array:-

```
int[][] a;
int [][]a;
int a[][];
int []a[];
```

Ex:-

```
class Test
{
public static void main(String[] args)
{
int[][] a={{10,20,30},{40,50,60}};
System.out.println(a[0][0]); //10
System.out.println(a[1][0]); //40
System.out.println(a[1][1]); //50
}
}
```

Methods (behaviors):-

1) Methods are used to provide the business logic of the project.

2) The methods like a functions in C-language called functions, in java language is called methods.

3) Inside the class it is possible to declare any number of methods based on the developer requirement.

4) As a software developer while writing method we have to fallow the coding standards like the method name starts with lower case letters if the method contains two words every inner word also starts uppercase letter.

5) It will improve the reusability of the code. By using methods we can optimize the code.

Syntax:-

[modifiers-list] return-Type Method-name (parameter-list) throws Exception

Ex:-

Public void m1()

Public void m2(int a,int b)

Method Signature:-

The name of the method and parameter list is called Method Signature. Return type and modifiers list not part of a method signature.

Ex:- m1(int a,int b)---- Method Signature

m2();-----------------   Method signature

There are 2 types of methods :

Instance methods – can be called using objects

Static methods – can be called directly

Ex:-

```
package com.madhu.basics;

public class MethodDemo {

void m1() // instance method

{

	System.out.println("m1 is instance method");
```

```
}
static void m2() // static method
{
	System.out.println("m2 is static method");
}
public static void main(String[] args) {
	MethodDemo methodDemo=new MethodDemo();
	methodDemo.m1(); // called using object
	m2(); // called directly
}
}
```

There are 2 types of instance methods :

1) Accessor methods just used to read the data. To read the reading the data use getters methods.

2) Mutator methods used to store the data and modify the data for the storing of data use setters methods.

```
class Test
{
String name;
int id;
//mutator method we are able to access and the data
void setName(String name)
{
this.name=name;
}
void setId(int id)
{
this.id=id;
```

```
}

//accessor methods are used to read the data

String getName()

{

return name;

}

int getId()

{

return id;

}

public static void main(String[] args)

{

Test t=new Test();

t.setName("Madhu Kumar Vundavalli");

t.setId(101);

String name=t.getName();

System.out.println(name);

int id=t.getId();

System.out.println(id);

}

}
```

Constructors :-

1) Constructors are executed as part of the object creation.

2) If we want to perform any operation at the time of object creation the suitable place is constructor.

3) Inside the java programming the compiler is able to generate the constructor and user is able to declare the constructor. so the constructors are provided by compiler and user.

There are two types of constructors :

1) Default Constructor.

a. Zero argument constructor.

2) User defined Constructor

a. zero argument constructor

b. parameterized constructor

Default Constructor:-

1) in the java programming if we are not providng any constructor in the class then compiler provides default constructor.

2) The default constructor is provided by the compiler at the time of compilation.

3) The default constructor provided by the compiler it is always zero argument constructors with empty implementation.

4) The compiler generated default constructor is executed by the JVM at the time of execution.

User defined constructors:-

Based on the user requirement user can provide zero argument constructor as well as parameterized constructor.

Rules to declare a constructor:-

1) Constructor name must be same as its class name.

2) Constructor doesn’t have no explicit return type if we are providing return type we are getting any compilation error and we are not getting any runtime errors just that constructor treated as normal method.

3) In the class it is possible to provide any number of constructors.

Example program for default constructor, parameterized constructor and constructor chaining :

```
package com.madhu.constructors;

public class Test {

	Test()

	{

		System.out.println("Default constructor");

	}
```

```
	Test(int a, int b)
	{
		this(); // calling constructor
		System.out.println("parameterised constructor");
		int c = a + b;
		System.out.println("a+b value is" + c);
	}
	public static void main(String[] args) {
		Test t=new Test(12, 14);
	}
}
```

Object oriented concepts :

Inheritance : The process of getting properties(variables) and behaviours(methods) from one class to another class is called inheritance.

The main purpose of the inheritance is code extensibility whenever we are extending automatically the code is reused.

In inheritance one class giving the properties and behavior and another class is taking the properties and behavior.

Inheritance is also known as is-a relationship means two classes are belongs to the same hierarchy.

By using extends keyword we are achieving inheritance concept.

In the inheritance the person who is giving the properties is called parent the person who is taking the properties is called child.

To reduce length of the code and redundancy of the code sun peoples introducing inheritance concept.

Types of inheritance :

Single inheritance : Extending the properties from single parent class to single child class is known as single inheritance.

Ex:-

```
class Parent
{
```

```
void m1()

{

System.out.println(“Parent class method”);

}

}

Class Child extends Parent

{

void m2()

{

System.out.println(“child class method”);

}

Public static void main(String args[])

{

Child c=new Child();

c.m1(); // called parent class method

c.m2(); // called child class method

}

}
```

Multi level inheritance : one super and one sub class at each and every level.

Multiple inheritance : The process of getting properties and behaviors form more than one super class to the one child class. The multiple inheritance is not possible in the java language using classes and it is supported using interfaces. so one class can extends only one class at time it is not possible to extends more than one class at time.

Hierarchical inheritance:- The process of getting properties and behaviors from one super class to the more than one sub classes is called hierarchical inheritance.

Hybrid inheritance:- Combination of any two inheritances is called as hybrid inheritance. If are taking the multilevel and hierarchical that combination is called hybrid inheritance.

Important points for inheritance :

Every class in the java programming is a child class of Object.

The root class for all java classes is Object class.

The default package in the java programming is java.lang package.

Polymorphism : polymorphism means many forms

Polymorphism is a Greek word poly means many and morphism means forms.

There are 2 types of polymorphism.

Compile time polymorphism (Early binding)

Run time polymorphism(Late binding)

Method Overloading:-

1) Two methods are said to be overloaded methods if and only if two methods are having same name but different argument list.

2) We can overload the methods in two ways in java language

a. Provide the different number of arguments to the same methods.

b. Provide the same number of arguments with different data types.

3) If we want achieve overloading one class is enough.

4) It is possible to overload any number of methods.

Eg:-

```
package com.madhu.oops;

public class MethodOverLoadingDemo {

	void add()

	{

		System.out.println("method with no args");

	}

	void add(int a)

	{

		System.out.println("Method with single argument :"+a);

	}

	void add(int a, int b)

	{
```

```
		int c=a+b;
		System.out.println("method with 2 arguments :"+(c));
	}
	public static void main(String[] args) {
		MethodOverLoadingDemo m=new MethodOverLoadingDemo();
		m.add();
		m.add(12);
		m.add(12, 14);
	}
}
```

Can we overload main method ?? yes.. we can overload main method

Eg:-

```
package com.madhu.oops;
class OverLoadMain
{
public static void main(int a)
{
System.out.println("integer parameter argument");
System.out.println(a);
}
public static void main(char ch)
{
System.out.println("character paramer argument");
System.out.println(ch);
}
public static void main(String[] args)
{
System.out.println("String[] parameter main method start");
```

```
main(100);

main('r');

}

}
```

Method Overriding :-

1) If the child class not satisfy the parent class method implementation then it is possible to override that method in the child class based on child class requirement.

2) If we want to achieve method overriding we need two class(child and parent).

3) In the overriding concept the child class and parent class method signatures must be same otherwise we are getting compilation error.

Parent class method is called over ridden method

Child class method is called over riding method

Eg:-

```
package com.madhu.oops;

public class Sample {

void show()

{

	System.out.println("parent class method");

}

}

package com.madhu.oops;

public class Sample1 extends Sample{

void show()

{

	super.show();

	System.out.println("child class method");
```

```
}

public static void main(String[] args) {

       Sample1 s=new Sample1();

       s.show();

}

}
```

Super keyword : if we have same method name with same number of arguments in parent class and child class, if we call the method by using child class object child class method will be called. In order to call parent class method we have to use super keyword.

Abstraction:-

Hiding the internal implementation and highlighting the set of services that process is called abstraction.

Ex:-

a. Bank ATM Screens (Hiding thee internal implementation and highlighting set of services like withdraw, money transfer, mobile registration).

b. Mobile phones (The mobile persons are hiding the internal circuit implementation and highlighting touch screen).

c. Syllabus copy (the institutions persons just highlighting the set of contents that persons provided the persons are not highlighting the whole content).

Ex:- Abstract classes

     Interfaces

The way of representation the methods are divided into two types

1) Normal methods

2) Abstract methods

Normal methods:-

Normal method is a method which contains declaration as well as implementation.

Ex:-

```
Void m1()

{

---------
```

--------body;

}

Abstract methods:-

The method which is having declaration but not implementations such type of methods are called abstract Method. Hence every abstract method should end with “;”.

The child classes are responsible to provide implementation for parent class abstract methods.

Ex:- void m1 (); -- abstract method

Based on above representation of methods the classes are devided into two types

1) Normal classes

2) Abstract classes

Normal classes:-

Normal class is a java class it contains only normal methods.

Abstract class:- Abstract class is a jav class which contains at least one abstract method.To specify the particular class is abstract and particular method is abstract method to the compiler use abstract modifier.

For the abstract classes it is not possible to create an object. Because it contains the un implemented methods.

For any class if we don’t want instantiation then we have to declare that class as abstract i.e., for abstract classes instantiation (creation of object) is not possible.

Eg:-

```
package com.madhu.oops;

public abstract class Bank1 {

void deposit()

{

       System.out.println("we can deposit upto 5 lakhs");

}

void withDrawl()

{
```

```
	System.out.println("we can with draw upto 2 lakhs");
}


abstract void loan();
abstract void interest();
}
package com.madhu.oops;
public class Bank2 extends Bank1 {
	void loan()
	{
	System.out.println("we can take loan upto 1 lakh");
	}
	void interest()
	 {
		System.out.println("interest is 2%");
	}
	public static void main(String[] args) {
		Bank2 b=new Bank2();
		b.deposit();
		b.withDrawl();
		b.loan();
		b.interest();
	}
}
```

if the child class is unable to provide the implementation for parent class abstract methods at that situation we can declare that class is an abstract then take one more child class in that class provide the implementation for remaining methods.

Even though class does not contain any abstract method still we can declare the class as abstract i.e. abstract class can contain zero number of abstract methods. The abstract classes it is not possible to create object.

Interfaces :

Interface is also one of the type of class it contains only abstract methods.

For the interfaces also .class files will be generated.

Each and every interface by default abstract hence it is not possible to create an object.

Interfaces not alternative for abstract class it is extension for abstract classes.

100 % pure abstract class is called interface.

The Interface contains only abstract methods means unimplemented methods.

Interfaces giving the information about the functionalities it are not giving the information about internal implementation.

To provide implementation for abstract methods we have to take separate class that class we can called it as implementation class for that interface.

Interface can be implemented by using implements keyword.

For the interfaces also the inheritance concept is applicable.

Eg:-

```
package com.madhu.oops;

public interface Bank3

{

void deposit();

void withDrawl();

void loan();

void interest();

}

package com.madhu.oops;

public class Bank4 implements Bank3

{

public void deposit()
```

```
{
System.out.println("we cand deposit upto 5 lakhs");
}
public void withDrawl()
{
System.out.println("we can with draw upto 2 lakhs");
}
public void loan()
 {
System.out.println("we can take loan upto 1 lakh");
}
public void interest()
{
System.out.println("interest is 2%");
}
public static void main(String[] args)
{
Bank4 b=new Bank4();
b.deposit();
b.withDrawl();
b.loan();
b.interest();
}
}
```

Encapsulation:-

The process of binding the data and code as a single unit is called encapsulation.

We are able to provide more encapusulation by taking the private data(variables) members. To get and set the values from private members use getters and setters to set the data and to get the data.

Ex:-

```
class Encapsulation

{

private int sid;

private int sname;

public void setSid(int x)

{

this.sid=sid;

}

public int getSid()

{

return sid;

}

public void setSname(String sname)

{

this.sname=sname;

}

public String getSname()

{

return sname;

}

}
```

To access encapsulated use fallowing code:-

```
class Test

{
```

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

{

Encapsulation e=new Encapsulation();

e.setSid(100);

e.setSname("ratan");

int num=e.getSid();

String name=e.getSname();

System.out.println(num);

System.out.println(name);

}

}
```

Strictfp:-

Strictfp is a modifier applicable for classes and methods.

If a method is declared as strictfp all floating point calculations in that method will follow IEEE754 standard. So that we will get platform independent results.

If a class is declared as stictfp then every concrete method in that class will follow IEEE754 standard so we will get platform independent results.

Native:-

Native is the modifier only for methods but not for variables and classes.

The native methods are implemented in some other languages like C and C++ hence native methods also known as “foreign method”.

For native methods implementation is always available in other languages and we are not responsible to provide implementation hence native method declaration should compulsory ends with “;”.

Ex:- public native String intern();

Public static native void yield();

Public static native void sleep(long);

Which of the fallowing declarations are valid:-

public static void main(String[] args)-------valid

public static void main(String... a)---------valid

public static int main(String... args)-------invalid

static public void mian(String a[])----------valid

final public static void mian(String[] madhu)---valid

final strictfp public static void main(String[] ramu)-----valid

final strictfp synchronized public static void main(String... veeru)------- valid

Scanner :-

It is a class present in java.util package.

It is used to take the values from the key board.

To get the integer value from the keyboard-------:- nextInt()

To get the String value from the keyboard---------:- next()

To get the floating values from the keyboard------:- nextFloat();

Eg :-

```
import java.util.*;
class Test
{
String ename;
int eid;
double esal;
int eage;
void details()
{
Scanner s=new Scanner(System.in);
System.out.println("enter emp name");
String ename=s.next();
this.ename=ename;
System.out.println("enter emp id");
int eid=s.nextInt();
this.eid=eid;
```

```
System.out.println("enter emp sal");
double esal=s.nextDouble();
this.esal=esal;
System.out.println("enter age");
int eage=s.nextInt();
this.eage=eage;
}
void display()
{
System.out.println("************emp details***********");
System.out.println(ename);
System.out.println(eid);
System.out.println(esal);
System.out.println(eage);
}
void status()
{
if (eage>40)
{
System.out.println("not elgible");
}
else
{
System.out.println("elgible");
}
}
public final strictfp synchronized static void main(String[] args)
{
```

```
Test t=new Test();

t.details();

t.display();

t.status();

}

}
```

Packages :-

1) The package contains group of related classes and interfaces.

2) The package is an encapsulation mechanism it is binding the related classes and interfaces.

3) We can declare a package with the help of package keyword.

4) Package is nothing but physical directory structure and it is providing clear-cut separation between the project modules.

5) Whenever we are dividing the project into the packages(modules) the sharability of the project will be increased.

Syntax:-

```
Package package_name;
```

Ex:- package com.dss;

The packages are divided into two types

1) Predefined packages

2) User defined packages

Predefined packages:-

The java predefined packages are introduced by sun peoples these packagescontains predefined classes and interfaces.

Ex:- java.lang

Java.io

Java.awt

Java.util

Java.net……..etc

Java.lang:-

The most commonly required classes and interfaces to write a sample program is encapsulated into a separate package is called java.lang package.

` Ex:- String(class)

       StringBuffer(class)

       Object(class)

       Runnable(interface)

       Cloneable(nterface)

Note:-

the default package in the java programming is java.lang if we are importing or not importing by default this package is available for our programs.

Java.io package:-

The classes which are used to perform the input output operations that are present in the java.io packages.

Ex:- FileInputStream(class)

     FileOutputStream(class)

     FileWriter(class)

     FileReader(class)

Java.net package :-

The classes which are required for connection establishment in the network that classes are present in the java.net package.

Ex:- Socket

     ServerSocket

     InetAddress

     URL

Java.awt package:-

The classes which are used to prepare graphical user interface those classes are present in the java.awt package.

Ex:- Button(class)

     Checkbox(class)

Choice(Class)

List(class)

User defined packages:-

1) The packages which are declared by the user are called user defined packages.

2) In the single source file it is possible to take the only one package. If we are trying to take two packages at that situation the compiler raise a compilation error.

3) In the source file it is possible to take single package.

4) While taking package name we have to fallow some coding standreds.

Whenever we taking package name don’t take the names like pack1, pack2, madhu, ramu…… these are not a proper coding formats.

If it is a predefined package or user defined package whenever we are using that package classes into our program we must make available that package into our program with the help of import statement.

Access Modifiers :-

Public :-

This is the modifier applicable for classes, methods and variables (only for instance and static variables but not for local variables).

If a class is declared with public modifier then we can access that class from anywhere (within the package and outside of the package).

If we declare a member(variable) as a public then we can access that member from anywhere but Corresponding class should be visible i.e., before checking member visibility we have to check class visibility.

Default :-

This is the modifier applicable for classes, methods and variables (only for instance and static variables but not for local variables).

If a class is declared with <default> modifier then we can access that class only within that current package but not from outside of the package.

Default access also known as a package level access.

The default modifier in the java language is default.

Private :-

private is a modifier applicable for methods and variables.

If a member declared as private then we can access that member only from within the current class.

If a method declare as a private we can access that method only within the class. it is not possible to call even in the child classes also.

Protected :-

If a member declared as protected then we can access that member with in the current package anywhere but outside package only in child classes.

But from outside package we can access protected members only by using child reference. If we try to use parent reference we will get compile time error.

Members can be accesses only from instance area directly i.e., from static area we can’t access instance members directly otherwise we will get compile time error.

String :-

String is a final class it is present in java.lang package.

String is nothing but a group of characters or character array.

Once we are creating String object it is not possible to do the modifications on existing object called immutability nature.

In String class .equals() is used for content comparision.

Eg :-

```
package com.madhu.strings;

public class StringDemo

{

public static void main(String[] args)

 {

String s="madhu";

System.out.println(s);

s.concat("kumar");

System.out.println(s);

String s1=s.concat("kumar");
```

```
System.out.println(s1);

System.out.println(s1.length());

System.out.println(s1.getClass());

System.out.println(s1.toLowerCase());

System.out.println(s1.toUpperCase());

System.out.println(s1.hashCode());

}

}
```

StringBuffer :-

String Buffer is a class present in the java.lang package.

StringBuffer is a final class so it can’t be inharited.

StringBuffer is a mutable class so it is possible to change the content in the same location.

StringBuffer .equals() method is used for reference comparision.

StinrgBuffer is mutable:-

Once we are creating a StringBuffer Object it is possible to the modification on existing object is called mutability nature.

Ex:-

```
class Test

{

public static void main(String[] args)

{

StringBuffer s1=new StringBuffer("rattaiah");

s1.append("addanki");//mutability

System.out.println(s1);

StringBuffer s2=new StringBuffer("durgasoft");

StringBuffer s3=s1;

System.out.println(s1.equals(s2));
```

```
System.out.println(s1.equals(s3));

System.out.println(s1==s2);

System.out.println(s1==s3);

}

}

class Test

{

public static void main(String[] args)

{

Test t1=new Test();

Test t2=t1;

Test t3=new Test();

System.out.println(t1.equals(t2)); //true

System.out.println(t1.equals(t3)); //false

String str1="ratan";

String str2="ratan";

System.out.println(str1.equals(str2)); //true

StringBuffer sb1=new StringBuffer("madhu");

StringBuffer sb2=new StringBuffer("madhu");

StringBuffer sb3=sb2;

System.out.println(sb1.equals(sb2)); //false

System.out.println(sb2.equals(sb3)); //true

}

}
```

StringBuilder :-

Introduced in jdk1.5 version.

StringBuilder is identical to StringBuffer except for one important difference.

Every method present in the StringBuilder is not Synchronized means that is not read safe.

multiple threads are allow to operate on StringBuilder methods hence the performance of the application is increased.

Cloneable :-

The process of creating exactly duplicate object is called cloning.

We can create a duplicate object only for the cloneable classes .

We can create cloned object by using clone()

The main purpose of the cloning is to maintain backup.

Eg :-

```
class Test implements Cloneable

{

int a=10;

int b=20;

public static void main(String[] args) throws CloneNotSupportedException

{

Test t1=new Test();

Test t2=(Test)t1.clone();

t1.a=100;

t1.b=200;

System.out.println(t1.a+"----"+t1.b);

System.out.println(t2.a+"----"+t2.b);

}

}
```

Wrapper classes :-

To represent primitive data types as a Object form we required some classes these classes are called wrapper classes.

All wrapper classes present in the java.lang package.

int,byte…. Acts as a primitives we can make the primitives into the objects is called wrapper calless the the wrapper classes are Integer,Short-----.

We are having 8 primitive data types hence sun peoples are providing 8 wrapper classes.

Byte, Short, Integer, Long, Float, Double these are child classes of Number class.

Primitives and their corresponding classes :-

Primitives	Wrapper classes
Byte	Byte
Int	Integer
Short	Short
Long	Long
Float	Float
Double	Double
Char	Character
Boolean	Boolean

Eg :-

```
class Test

{

public static void main(String[] args)

{

float f=10.7f;

Float f1=new Float(f);

System.out.println(f1);

Float f2=new Float("10");

System.out.println(f2);

Float f3=new Float("ten");

System.out.println(f3); //NumberFormatException

Integer i1=new Integer(2);

System.out.println(i1);

Integer i2=new Integer("two");

System.out.println(i2); //NumberFormatException

}
```

```
}

class Test

{

public static void main(String[] args)

{

int a=10; //primitive variable

System.out.println(a);

System.out.println(a.toString()); //CE:-int cant be dereferenced

Integer i=new Integer("100"); //reference variable

System.out.println(i);

System.out.println(i.toString());

}

}
```

The main importance of wrapper classes :-

To convert a data types into the object means we are giving object from data types by using constructor.

To convert String into the data types by using parsexxx() method

Utility methods:-

1. valueOf()

2. xxxValue()

3. parsexxx()

4. toString()

1) valueOf():-

By using valueof() we are creating wrapper object and it is a alternative to the constructor.

Ex :-

```
class Test

{
```

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

{

//by using constructor converting String/primitive to wrapper object

Integer i=new Integer(10);

System.out.println(i);

//by using valueOf() converting String/primitive to the wrapper object

Boolean b=Boolean.valueOf("true");

System.out.println(b);

}

}
```

XxxValue():-

by using XXXValue() method we are converting wrapper objects into the corresponding primitive values.

Ex :-

```
class Test

{

public static void main(String[] args)

{

Integer i=Integer.valueOf(150);

System.out.println("byte value :"+i.byteValue()); //-106

System.out.println("short value :"+i.shortValue()); //150

System.out.println("int value :"+i.intValue()); //150

System.out.println("long value :"+i.longValue()); //150

System.out.println("float value :"+i.floatValue()); //150.0

System.out.println("double value :"+i.doubleValue()); //150.0

Character c=new Character('s');

char ch=c.charValue();

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

```
Boolean b=new Boolean(false);

boolean bb=b.booleanValue();

System.out.println(bb);

}

}
```

parseXXX():-

by using above method we are converting String into the corresponding primitive.

Ex :-

```
class Test

{

public static void main(String[] args)

{

String str1="10";

String str2="20";

System.out.println(str1+str2); //1020

int a=Integer.parseInt(str1);

float f=Float.parseFloat(str2);

System.out.println(a+f); //30.0

}

}
```

Autoboxing and Autounboxing :- (introduced in the 1.5 version)

Until 1.4 version we are not allowed to place primitive in the palc wrapper and wrapper in the place of primitive. The programmer is responsible person to do the explicit conversion primitive to the wrapper and wrapper to the primitive.

Autoboxing :-

```
Integer i=10;

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

The above statement does not work on the 1.4 and below versions. The auto conversion of the primitive into the Wrapper object is called the autoboxing these conversions done by compiler at the time of compilation.

Autounboxing:-

```
int a=new Integer(100);

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

The auto conversion of the wrapper object to the primitive value is called autounboxing and these conversions are done by compiler at the time of compilation.

Ex :-

```
class Test

{

static Integer i=10; //i is wrapper object

static int j; //j is primitive variable

static void print(int i)

{

j=i;

System.out.println(j);

}

public static void main(String[] args)

{

print(i);

System.out.println(j);

}

}
```

Files :- (java.io package)

Java.io is a package which contains number of classes by using that classes we are able to send the data from one place to another place.

In java language we are transferring the data in the form of two ways:-

1. Byte format

2. Character format

Streams are two types:-

1. Byte oriented streams.(supports byte formatted data to transfer)

2. Character oriented stream.(supports character formatted data to transfer)

Byte oriented streams:-

Java.io.FileInputStream

To read the data from the destination file to the java application we have to use FileInputSream class.

To read the data from the .txt file we have to read() method.

Java.io.FileOutputStream:-

To write the data to the destination file we have to use the FileOutputStream.

To write the data to the destination file we have to use write() method.

Ex:- it will supports one character at a time.

```
import java.io.*;

class Test

{

static FileInputStream fis;

static FileOutputStream fos;

public static void main(String[] args)

{

try

{

fis=new FileInputStream("get.txt");

fos=new FileOutputStream("set.txt",true);

int c;

while ((c=fis.read())!=-1)

{

fos.write(c);
```

```
}

fis.close();

fos.close();

}

catch(IOException io)

{

System.out.println("getting IOException");

}

}

}
```

BufferedReader:- to read the data line by line format and we have to use readLine() to read the data.

PrintWriter :- to write the data line by line format and we have to use println() to write the data.

```
import java.io.*;

class Test

{

static BufferedReader br;

static PrintWriter pw;

public static void main(String[] args)

{

Try

{

br=new BufferedReader(new FileReader("get.txt"));

pw=new PrintWriter(new FileWriter("set.txt"));

String line;

while ((line=br.readLine())!=null)

{
```

```
pw.println(line);

}

br.close();

pw.close();

}

catch(IOException io)

{

System.out.println("getting IOException");

}

}

}
```

Buffered Streams:-

Up to we are working with non buffered streams these are providing less performance because these are interact with the hard disk, network.

Now we have to work with Buffered Streams

BufferedInputStream read the data from memory area known as Buffer.

We are having four buffered Stream classes

1. BufferedInputStream
2. BufferedOutputStream
3. BufferedReader
4. BufferedWriter

Ex:-

```
import java.io.*;

class Test

{

static BufferedReader br;

static BufferedWriter bw;

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

```
{
Try
{
br=new BufferedReader(new FileReader("Test1.java"));
bw=new BufferedWriter(new FileWriter("States.java"));
String str;
while ((str=br.readLine())!=null)
{
bw.write(str);
}
br.close();
bw.close();
}
catch(Exception e)
{
System.out.println("getting Exception");
}
}
}
```

Serialization:-

The process of saving an object to a file (or) the process of sending an object across the network is called serialization.

But strictly speaking the process of converting the object from java supported form to the network supported form of file supported form.

To do the serialization we required fallowing classes

1. FileOutputStream
2. ObjectOutputStream

Deserialization:-

The process of reading the object from file supported form or network supported form to the java supported form is called deserialization.

We can achieve the deserialization by using fallowing classes.

1. FileInputStream

2. ObjectInputStream

Ex:-

```
import java.io.Serializable;

public class Student implements Serializable

{

int id;

String name;

int marks;

public Student(int id, String name,int marks)

{

this.id = id;

this.name = name;

this.marks=marks;

}

}
```

To perform serialization :- we are writing the object data to the file called abc.txt file we are transferring that file across the network.

```
import java.io.*;

class Serializable1

{

public static void main(String args[])throws Exception

{

Student s1 =new Student(211,"ravi",100);
```

```
FileOutputStream fos=new FileOutputStream("abc.txt",true);

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(s1);

oos.flush();

System.out.println("Serializable process success");

}

}
```

To perform deserialization:- in the network the file is available with java data to read the data we have to go for deserialization.

```
import java.io.*;

class Deserialization

{

public static void main(String args[])throws Exception

{

//deserialization process

FileInputStream fis=new FileInputStream("abc.txt");

ObjectInputStream ois=new ObjectInputStream(fis);

Student s=(Student)ois.readObject();

System.out.println("the student name is:"+s.name);

System.out.println("the stuent id is:"+s.id);

System.out.println("the student marks:"+s.marks);

System.out.println("deserialization success");

}

}
```

Transient :- Transient modifier is the modifier applicable for only variables and we can’t apply for methods and classes.

At the time of serialization, if we don’t want to save the values of a particular variable to meet security constraints then we should go for transient modifier.

At the time of serialization JVM ignores the original value of transient variable and default value will be serialized.

Eg :-

```
import java.io.*;

import java.io.Serializable;

class Student implements Serializable

{

transient int id=100;

transient String name="ravi";

}

class Serializable1

{

public static void main(String args[])throws Exception

{

 Student s1=new Student();

System.out.println("the stuent id is:"+s1.id);

System.out.println("the student name is:"+s1.name);

FileOutputStream fos=new FileOutputStream("ratan.txt",true);

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(s1);

FileInputStream fis=new FileInputStream("ratan.txt");

ObjectInputStream ois=new ObjectInputStream(fis);

Student s=(Student)ois.readObject();

System.out.println("the stuent id is:"+s.id);

System.out.println("the student name is:"+s.name);

}

}
```

Exception Handling :-

Dictionary meaning of the exception is abnormal termination.

An exception is a problem occurred during execution time of the program.

An unwanted unexpected event that disturbs normal flow of execution called exception.

Exception is nothing but a object.

Exception is a class present in java.lang package.

All the exceptions are nothing but objects called classes.

Whenever user is entered invalid data then Exception is occur.

A file that needs to be opened can’t found then Exception is occurred.

Exception is occurred when the network has disconnected at the middle of the communication.

Types of Exceptions:-

As per sun micro systems standards The Exceptions are divided into three types

1) Checked Exception

2) Unchecked Exception

3) Error

Checked Exception :-

The Exceptions which are checked by the compiler at compilation time for the proper execution of the program at runtime is called CheckedExceptions.

Ex :- IOException,SQLException etc.

Some of the checked Exceptions in the java language :-

Exception	Description
ClassNotFoundException	If the loaded class is not available
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread
NoSuchMethodException	If the requested method is not available

UncheckedException :-

The exceptions which are not checked by the compiler at compilation time is called uncheckedException. These checking down at run time only.

Ex:- ArithmeticException, NullPointerException, etc.

Some of the unchecked Exceptions in the java language :-

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.(out of range)
ClassCastException	If the conversion is Invalid.
IllegalArgumentException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NullPointerException	Invalid use of a null reference.

Error :-

Errors are caused by lack of system resources . these are non recoverable.

Ex:- StackOverFlowError,OutOfMemoryError,AssertionError etc……

Good point:-

The Exception whether it is checked or unchecked the Exceptions are occurred at runtime.

Difference between Exception and Error :-

Exception:-

An exception is unwanted unexpected event these are caused by the programmer mistake. Exceptions are recoverable.

Ex:- IOException,SQLExcetion,RuntimeExecption etc.

Error :-

Errors are caused by lack of system resources . these are non recoverable.

Ex :- StackOverFlowError,AssertionError etc.

Note :-

1) RuntimeException and its child classes and Error and its child classes are Unchecked remaining all are checkedExceptions.

2) Root class for all Exception hierarchy is Throweable class.

In java Exception handling we are having 5 key words:-

1) try

2) catch

3) finally

4) throw

5) throws

Exception Handling :-

It is recommended to handle the Exception the main of the Exception Handling is normal Execution of the program or graceful termination of the program at runtime.

We are able to handle the exception in two ways.

1. By using try-catch blocks

2. By using throws keyword.

Exception handling by using Try –catch block :-

1) In java language we are handling the exceptions By using try and catch blocks. try block contains risky code of the program and catch block contains handling code of the program.

2) Catch block code is a alternative code for Exceptional code. If the exception is raised the alternative code is executed fine then rest of the code is executed normally.

Syntax :-

```
try

{

Risky code

}

Catch(ExceptionName reference_variable)

{

Risk handling code

}
```

Before try and catch :- The program goes to abnormal termination

```
class Test
{
public static void main(String[] args)
{
System.out.println("Techfort");
System.out.println("software services pvt ltd");
System.out.println(10/0);
System.out.println("solutions");
}
}
```

Output :- Techfort

Software

Exception in Thread “main” :java.lang.ArithmeticException: / by zero

After try catch :-

1) If we are taking try-catch the program goes to normal termination. Because the risky code we are taking inside the try block and handling code we are taking inside the catch block.

2) If the exception is raised in the try block the corresponding catch block is executed.

3) If the corresponding catch block is not there program goes to abnormal termination.

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

```
{
System.out.println(10/0);
}
catch (ArithmeticException e)
{
System.out.println("you are getting AE "+e);
}
System.out.println("solutions");
}
}
```

Output:- Techfort

Software

You are getting AE: java.lang.ArithmeticException: / by zero

Solutions.

Ex :-

Exception raised in try block the JVM will search for corresponding catch block if the catch block is matched, corresponding catch block will be executed and rest of the code is executed normally.

```
class Test
{
public static void main(String[] args)
{
System.out.println("program starts");
try
{
int[] a={10,20,30};
System.out.println(a[0]);
System.out.println(a[1]);
```

```
System.out.println(a[2]);

System.out.println(a[3]);

}

catch(ArrayIndexOutOfBoundsException ae)

{

System.out.println("we are getting exception");

}

System.out.println("rest of the code");

}

}
```

Finally block :-

1) finally is a block it is always executed irrespective of try and catch.

2) Finally contains clean-up code.

3) It is not possible to write finally alone . we must take try-catch-finally otherwise take the try-finally these two are the possibilities. If we are taking any other we are getting compilation error saying finally without try block .

Syntax :-

```
try

{

risky code;

}

catch (Exception obj)

{

handling code;

}

finally

{

free code;
```

```
}
```

Ex :- Exception raised in try block and corresponding catch block is matched then rest of the code is executed normally.

```
class Test

{

public static void main(String[] args)

{

try

{

System.out.println("Techfort");

System.out.println(10/0);

}

catch (ArithmeticException ae)

{

System.out.println("u r getting ae:"+ae);

}

finally

{

System.out.println("finally block is always executed");

}

System.out.println("rest of the code");

}

}
```

Output :- Techfort

U r getting ae:ArithmeticException : /by zero

Finally block is always executed

Throw :-

1) The main purpose of the throw keyword is to creation of Exception object explicitly either for predefined or user defined .

2) Throw keyword works like a try block. The difference is try block is automatically find the situation and creates a Exception object implicitly. Whereas throw keyword creates a Exception object explicitly.

Ex :-

```
class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch (ArithmeticException ae)

{

System.out.println("we are getting Exception "+ae);

}

}

}
```

Output :- we are getting Exception ArtithmeticException: / by zero

Note :-

In the above program the main method is responsible to create a exception object. So the main method is creating exception object implicitly. The programmer is not responsible person to creates a exception object.

Ex :-

```
class Test

{

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

```
{

throw new ArithmeticException("we are getting Exception / by zero man");

}

}
```

Output:-

```
Exception in Thread “main” :java.lang.ArithmeticException:we are getting Exception/
by zero man
```

Throws :-

1) Throw keyword is used to create exception object explicitly. But the main purpose of the throws keyword is bypassing the generated exception from present method to caller method.

2) Throw keyword is used in the method body. But throws keyword we have to use in the method declaration.

3) It is possible to throws any number of exceptions at a time based on the programmer requirement.

In the java language we are handling the Exception in two ways :-

1) By using try-catch blocks

2) By using throws keyword

By using try-catch blocks:-

Ex :-

```
import java.io.*;

class Student

{

void studentDetails()

{

try

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("please enter student name");

String sname=br.readLine();
```

```
System.out.println(“u r name is:”+sname);
}
catch(IOException e)
{
System.out.println("we are getting Exception"+e);
}
}
public static void main(String[] args)
{
Student s1=new Student();
s1.studentDetails();
}
}
```

By using throws keyword :-

Ex 1:-

```
import java.io.*;
class Student
{
void studentDetails()throws IOException
{ BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("please enter student name");
String sname=br.readLine();
System.out.println("u r name is:"+sname);
}
public static void main(String[] args)throws IOException
{
Student s1=new Student();
s1.studentDetails();
```

```
}
}
```

Ex 2 :-

```
import java.io.*;
class Student
{
void studentDetails()throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("please enter student name");
String sname=br.readLine();
System.out.println("u r name is:"+sname);
}
public static void main(String[] args)
{
Student s1=new Student();
try
{
s1.studentDetails();
}
catch (IOException ie)
{
System.out.println("this is my handling code");
}
}
}
```

Exceptions :-

There are two types of exceptions present in the java language

1) Predefined Exceptions.

2) User defined Exceptions.

Predefined Exception:-

Predefined classes comes along with the software based on your requirement we have to create a objects.

Ex :- ArithmeticException,IOException,NullPointerException……..etc

User defined Exceptions:-

Based on the user requirement user can creates a Exception is called user defined Exception.

Ex:InvaliedAgeException,BombBlostException……..etc

To create user defined Exceptions:-

1) To create user defined exception we have to take an user defined class that is a sub class to the RuntimeException(for creation of unchecked Exceptions) .

2) To create user defined exception we have to take user defined class that is subclass to the Exception(for creation of checked Exceptions)

3) Each and every Exception contains two constructors

a) default constructor

b) parameterized constructor

Eg :-

```
import java.util.*;

class Test

{

static void validate(int age)

{

if (age<18)

{

throw new InvaliedAgeException("not elgible for vote");
```

```
}

else

{

System.out.println("welcome to the voteing");

}

}

public static void main(String[] args)

{

Scanner s=new Scanner(System.in);

System.out.println("please enter age");

int age=s.nextInt();

validate(age);

}

}
```

Different types of exceptions :-

ArrayIndexOutOfBoundsException:-

whenever we are calling array with out of range at that moment we are getting ArrayIndexOutOfBoundsException.

Ex :-

```
class Test

{

public static void main(String[] args)

{

try

{

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

System.out.println(a[0]);//10

System.out.println(a[1]);//20
```

```
System.out.println(a[2]);//30

System.out.println(a[3]);//ArrayIndexOutOfBoundsException

}

catch (ArrayIndexOutOfBoundsException ae)

{

System.out.println("boss u r geting ArrayIndexOutOfBoundsException");

System.out.println("check u r code once");

}

}

}
```

NumberFormatException :-

At the time of converting String value into the integer value we are getting NumberFormatException.

Ex:-

```
class Test

{

public static void main(String[] args)

{

try

{

String str="123";

int a=Integer.parseInt(str);

System.out.println(a);//conversion(string - int) is good

String str1="abc";

int b=Integer.parseInt(str1);

System.out.println(b);//NumberFormatException

}

catch (NumberFormatException ae)
```

```
{
System.out.println("boss u r geting NumberFormatException");
System.out.println("check once u r code");
}
}
}
```

NullPointerException :-

If we are having ‘null’ in any variable in that variable we are performing any operation we are getting NummpointerException.

Ex :-

```
class Test
{
public static void main(String[] args)
{
try
{
String str="rattaiah";
System.out.println(str.length());//8
String str1=null;
System.out.println(str1.length());//NullPointerException
}
catch (NullPointerException ne)
{
System.out.println("boss u r geting nullpointerexception");
System.out.println("check once u r code");
}
}
}
```

ArithmeticException :-

Whenever we are performing / by zero operation we are getting ArithmeticException.

Ex:-

```
class Test
{
public static void main(String[] args)
{
try
{
int a=10/2;
System.out.println(a);//5
```



```
int b=10/0;
System.out.println(b);//ArithmeticExceptiom
}
catch (ArithmeticException ne)
{
System.out.println("boss u r geting ArithmeticException");
System.out.println("check once u r code");
}
}
}
```

IllegalArgumentException :-

Thread priority range is 1-10

1--low priority

10--high priority

If we are giving priority out of range then we are getting IllegalArgumentException.

Ex :-

```
class Test
{
public static void main(String[] args)
{
Thread t=new Thread();
t.setPriority(11);//IllegalArgumentException
}
}
```

IllegalThreadStateException :-

Whenever we are trying to restart the already start thread then we are getting IllegalThreadStateException.

Ex :-

```
class Test
{
public static void main(String[] args)
{
Thread t=new Thread();
t.start();
t.start();//IllegalThreadStateException
}
}
```

Different types of Errors:-

StackOverflowError:-

Whenever we are calling method recursively then we are getting StackOverflowError.

Ex :-

```
class Test
{
```

```
void m1()
{
m2();
System.out.println("this is Rattaiah");
}
void m2()
{
m1();
System.out.println("from durgasoft");
}
public static void main(String[] args)
{
Test t=new Test();
t.m1();
}
}
```

OutOfMemoryError:-

If we are creating objects greater than the heap memory then we are getting OutOfMemoryError.

Ex :-

```
class Test
{
public static void main(String[] args)
{
int[] a=new int[100000000];//OutOfMemoryError
}
}
```

NoClassDefFoundError :-

Whatever the class if we want to execute if the class is not available at runtime we are getting NoClassDefFoundError.

Ex :-

```
class Test

{

public static void main(String[] args)

{

System.out.println("Madhu kumar vundavalli");

}

}
```

Output :- javac Test.java

Java Test

o/p:- Madhu kumar vundavalli

if we are executing class ABC (java ABC) if that class is not available then we are getting NoClassDefFoundError.

Multi Threading :

The earlier days the computer’s memory is occupied only one program after completion of one program it is possible to execute another program is called uni programming.

Whenever one program execution is completed then only second program execution will be started such type of execution is called co operative execution, this execution we are having lot of disadvantages.

a. Most of the times memory will be wasted.

b. CPU utilization will be reduced because only program allow executing at a time.

c. The program queue is developed on the basis co operative execution

To overcome above problem a new programming style will be introduced is called multiprogramming :-

1) Multiprogramming means executing the more than one program at a time.

2) All these programs are controlled by the CPU scheduler.

3) CPU scheduler will allocate a particular time period for each and every program.

4) Executing several programs simultaneously is called multiprogramming.

5) In multiprogramming a program can be entered in different states.

a. Ready state.

b. Running state.

c. Waiting state.

6) Multiprogramming mainly focuses on the number of programs.

Advantages of multiprogramming:-

1. CPU utilization will be increased.

2. Execution speed will be increased and response time will be decreased.

3. CPU resources are not wasted.

Thread :-

1) Thread is nothing but separate path of sequential execution.

2) The independent execution technical name is called thread.

3) Whenever different parts of the program executed simultaneously that each and every part is called thread.

4) The thread is light weight process because whenever we are creating thread it is not occupying the separate memory it uses the same memory. Whenever the memory is shared means it is not consuming more memory.

5) Executing more than one thread a time is called multithreading.

Single threaded model :-

```
class Test

{

public static void main(String[] args)

{

System.out.println("Hello World!");

System.out.println("hi Kodandarmu"); body

System.out.println("hello techfort");

}
```

} In the above program only one thread is available is called main thread to know the name of the thread we have to execute the fallowing code.

```
class Test

{

public static void main(String[] args)

{

System.out.println("Hello World!");

Thread t=Thread.currentThread();

System.out.println("currrent thread information is : "+t);//[main,5,main]

System.out.println("currrent thread priority is : "+t.getPriority());//5

System.out.println("currrent thread name is : "+t.getName());

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

System.out.println("hello durgasoft");

}

}
```

In the above program only one thread is available name of that thread is main thread.

The main important application areas of the multithreading are :-

1. Developing video games

2. Implementing multimedia graphics.

3. Developing animations

There are two different ways to create a thread is available

1) Create class that extending standered java.lang.Thread Class

2) Create class that Implementing java.lang.Runnable interface

First approach to create thread extending Thread class :-

Step 1:-

Creates a class that is extend by Thread classes and override the run() method

```
class MyThread extends Thread

{
```

```
public void run()
{
System.out.println("business logic of the thread");
System.out.println("body of the thread");
}
}
```

Step 2:-

Create a Thread object

```
MyThread t=new MyThread();
```

Step 3:-

Starts the execution of a thread.

```
t.start();
```

In this approach take one user defined class class that is extending Thread class .

Ex :-

```
class MyThread extends Thread
{
public void run()
{
System.out.println("Rattaiah from durgasoft");
System.out.println("body of the thread");
}
}
class ThreadDemo
{
public static void main(String[] args)
{
MyThread t=new MyThread();
t.start();
```

}

}

Note :-

1) Whenever we are calling t.start() method the JVM search for the start() in the MyThread class but the start() method is not present in the MyThread class so JVM goes to parent class called Thread class and search for the start() method.

2) In the Thread class start() method is available hence JVM is executing start() method.

3) Whenever the thread class start() that start() is responsible person to call run() method.

4) Finally the run() automatically executed whenever we are calling start() method.

5) Whenever we are giving a chance to the Thread class start() method then only a new thread will be created.

Life cycle stages are:-

1) New

2) Ready

3) Running state

4) Blocked / waiting / non-running mode

5) Dead state

New :-

MyThread t=new MyThread();

Ready :-

t.start()

Running state :-

If thread scheduler allocates CPU for particular thread. Thread goes to running state

The Thread is running state means the run() is executed.

Blocked State :-

If the running thread got interrupted of goes to sleeping state at that moment it goes to the blocked state.

Dead State :-

If the business logic of the project is completed means run() over thread goes dead state.

Second approach to create thread implementing Runnable interface:-

Step 1 :-

Creates a class that implements Runnable interface.

```
class MyClass extends Runnable

{

public void run()

{

System.out.println("Rattaiah from durgasoft");

System.out.println("body of the thread");

}

}
```

Step 2 :-

Creating a object.

```
MyClass obj=new MyClass();
```

Step 3 :-

Creates a Thread class object.

```
Thread t=new Thread(obj);
```

Step 4 :-

Starts the execution of a thread.

```
t.start();
```

implementing Runnable interface

```
class MyThread implements Runnable

{

public void run()

{

System.out.println("Rattaiah from durgasoft");
```

```
System.out.println("body of the thread");

}

}

class ThreadDemo

{

public static void main(String[] args)

{

MyClasss obj=new MyClass();

Thread t=new Thread(obj);

t.start();

}

}
```

Step 1:-

the Class MyClass implements the Runnable interface and overriding run() method and contains the logic associates with the body of the thread.

Step 2:-

Creates the object of implementation class this is not like a first mechanism.

Step 3 :-

Creates a generic thread object then pass the MyClass reference variable as a parameter to that object.

Step 4 :-

As a result of third step 3 a thread object is created in order to execute this thread method we need to class start() method. Then new thread is executed.

We are having two approaches :-

First approach:-

1) By extending the thread class, the derived class itself is a thread object and it gains full control over the thread life cycle.

2) Another important point is that when extending the Thread class, the sub class cannot extend any other base classes because Java allows only single inheritance.

if the program needs a full control over the thread life cycle, then extending the Thread class is a good choice.

Second approach :-

1) Implementing the Runnable interface does not give developers any control over the thread itself, as it simply defines the unit of work that will be executed in a thread.

2) By implementing the Runnable interface, the class can still extend other base classes if necessary.

if the program needs more flexibility of extending other base classes, implementing the

Runnable interface would be preferable.

Difference between t.start() and t.run() :-

In the case of t.start(), Thread class start() is executed a new thread will be created that is responsible for the execution of run() method.

But in the case of t.run() method, no new thread will be created and the run() is executed like a normal method call by the main thread.

If we are overriding start() method then JVM is executes override start() method at this situation we are not giving chance to the thread class start() hence n new thread will be created only one thread is available the name of that thread is main thread.

Getting and setting names of Thread:-

1) Every Thread in java has some name if may be default name provided by the jvm or customized name provided by the programmer.

The following methods are useful to set and get the name of a Thread.

a. Public final String getName()

b. Public final void setName(String name)

Ex :-

```
class MyThread extends Thread

{

}

class Test

{
```

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

{

System.out.println(Thread.currentThread().getName());

MyThread t=new MyThread();

System.out.println(t.getName());

Thread.currentThread().setName("madhu");

System.out.println(Thread.currentThread().getName());

}

}
```

Thread Priorities :-

Every Thread in java has some property. It may be default priority provided be the JVM or customized priority provided by the programmer.

The valid range of thread priorities is 1 – 10. Where one is lowest priority and 10 is highest priority.

The default priority of main thread is 5. The priority of child thread is inherited from the parent.

Thread defines the following constants to represent some standard priorities.

Thread Scheduler will use priorities while allocating processor the thread which is having highest priority will get chance first and the thread which is having low priority.

If two threads having the same priority then we can’t expect exact execution order it depends upon Thread Scheduler.

The thread which is having low priority has to wait until completion of high priority threads.

Three constant values for the thread priority.

a. MIN_PRIORITY = 1

b. NORM_PRIORITY = 5

c. MAX_PRIORITY = 10

Thread class defines the following methods to get and set priority of a Thread.

a. Public final int getPriority()

b. Public final void setPriority(int priority)

Here ‘priority’ indicates a number which is in the allowed range of 1 – 10. Otherwise we will get Runtime exception saying “IllegalArgumentException”.

Some of the thread class methods :-

Sleep() :-

The sleep() method causes the current thread to sleep for a specified amount of time in milliseconds.

public static void sleep(long millis) throws InterruptedException

public static void sleep(long millis,int nanosec) throws InterruptedException

For example, the code below puts the thread in sleep state for 5 minutes:

Ex :-

```
class Test

{

public static void main(String[] args)

{

try

{

for (int i=0;i<10 ;i++)

{

System.out.println("Rattaiah");

Thread.sleep(5*1000); //5 seconds

Thread.sleep(5*60*1000); // 5 minutes

}

}

catch (InterruptedException ie)

{

System.out.println("the thread is got innterupted");

}

}
```

```
}
```

Ex :-

```
class Test
{
public static void main(String[] args)throws InterruptedException
{
System.out.println("Rattaiah");
Thread.sleep(3*1000);
}
}
```

yield() :-

yield() method causes to pause current executing Thread for giving the chance for waiting threads of same priority.

If there are no waiting threads or all threads are having low priority then the same thread will continue its execution once again.

Syntax:-

```
Public static native void yield();
```

Ex:-

```
class MyThread extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
Thread.yield();
System.out.println("child thread");
}
}
```

```
}

class ThreadYieldDemo

{

public static void main(String[] args)

{

MyThread t1=new MyThread();

t1.start();

for(int i=0;i<10;i++)

{

System.out.println("main thread");

}

}

}
```

join() :-

If a Thread wants to wait until completing some other thread then we should go for join() method.

1. Public final void join()throws InterruptedExcetion

2. Public final void join(long ms)throws InterruptedException

3. Public final void join(long ms, int ns)throws InterruptedException

Ex :-

```
class MyThread extends Thread

{

public void run()

{

for (int i=0;i<5;i++ )

{

Try

{
```

```
System.out.println("rattaiah");
Thread.sleep(3*1000);}
catch(InterruptedException iee)
{
System.out.println("gettting innterupted exctpion");
}
}
}
}
class ThreadDemo
{
public static void main(String[] args)
{
MyThread t1=new MyThread();
MyThread t2=new MyThread();
t1.start();
try
{
t1.join();
}
catch (InterruptedException ie)
{
System.out.println("interrupted Exception");
}
t2.start();
}
}
```

Synchronized :-

Synchronized modifier is the modifier applicable for methods but not for classes and variables.

If a method or a block declared as synchronized then at a time only one Thread is allowed to operate on the given object.

The main advantage of synchronized modifier is we can resolve data inconsistency problems.

But the main disadvantage of synchronized modifier is it increases the waiting time of the Thread and effects performance of the system .Hence if there is no specific requirement it is never recommended to use.

The main purpose of this modifier is to reduce the data inconsistence problems.

Daemon threads:-

The threads wchich are executed at background is called daemon threads.

Ex:- garbage collector,ThreadSchedular.default exceptional handler.

Non-daemon threads:-

The threads which are executed fore ground is called non-daemon threads.

Ex :- normal java application.

Collection Frame Work :-

Limitations of arrays:-

1) Array is indexed collection o fixed number of homogeneous data elements

2) Arrays can hold homogeneous data only

3) Once we created an array no chance of increasing o decreasing size of array

Ex :-

Student[ ] s=new Student[100];

S[0]=new Student();

S[1]=new Student();

S[2]=new Customer();------compilation error

To overcome the above limitations of array the sun peoples are introduced collections concept

Collections:-

1) collection can hold both homogeneous data and heterogeneous data

2) collections are growable in nature

3) Memory wise collections are good. Recommended to use.

4) Performance wise collections are not recommended to use .

Collections :-

If we want to represent group of as a single entity then we should go for collection.

In the collection framework we having 9 key interfaces:-

1. Collection

2. List

3. Set

4. SortedSet

5. NavigablaSet

6. Queue

7. Map

8. SotedMap

9. NavigableMap

ArrayList:-

class ArrayList implements List

the collection classes stores only objects but we are passing primitives these primitives are automatically converts into objects is called autoboxing.

1) Introduced in 1.2 version.

2) ArrayList supports dynamic array that can be grow as needed.it can dynamically increase and decrease the size.

3) Duplicate objects are allowed.

4) Null insertion is possible.

5) Heterogeneous objects are allowed.

6) The under laying data structure is growable array.

7) Insertion order is preserved.

Ex:-

```
import java.util.*;

class ArrayListDemo

{

public static void main(String[] args)

{

//creation of ArrayList

ArrayList al=new ArrayList();

System.out.println("initial size of the arraylist:"+al.size());

//adding elements to the ArrayList

al.add("a");

al.add("A");

al.add("a");

al.add(null);

al.add(10);

al.add(1,"simon");

//print the ArrayList elements

System.out.println(al);

System.out.println("ArrayList size:"+al.size());

//remove the elements of ArrayList

al.remove("a");

System.out.println("ArrayList size:"+al.size());

System.out.println(al);

}

}
```

LinkedList :-

Class LinkedList implements List

1) Introduced in 1.2 v

2) Duplicate objects are allowed

3) Null insertion is possible

4) Heterogeneous objects are allowed

5) The under laying data structure is double linked list.

6) Insertion ode is preserved.

Ex :-

```
import java.util.*;
class Demo
{
public static void main(String[] args)
{
LinkedList ll=new LinkedList();
System.out.println(ll.size());
//add the elements to the LinkedList
ll.add("a");
ll.add(10);
ll.add(10.6);
ll.addFirst("ratan");
ll.addLast("anu");
System.out.println("original content :"+ll);
System.out.println(ll.size());
//remove elements from LinkedList
ll.remove(10.6);
ll.remove(0);
System.out.println("after deletion content :"+ll);
```

```
System.out.println(ll.size());

//remove first and last elements

ll.removeFirst();

ll.removeLast();

System.out.println("ll after deletion of first and last :"+ll);

//get and set a value

int a=(Integer)ll.get(0);

ll.set(0,a+"madhu");

System.out.println("ll after change:"+ll);

}

}
```

Vector:- (legacy class introduced in 1.0 version)

1) Introduced in 1.0 v legacy classes.

2) Duplicate objects are allowed

3) Null insertion is possible

4) Heterogeneous objects are allowed

5) The under laying data structure is growable array.

6) Insertion order is preserved.

7) Every method present in the Vector is synchronized and hence vector object is Thread safe.

Ex:-

```
import java.util.*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

for (int i=0;i<10 ;i++ )
```

```
{
v.addElement(i);
}
System.out.println(v);
v.addElement("veeresh");
System.out.println(v);
v.removeElement(0);
System.out.println(v);
v.clear();
System.out.println(v);
}
}
```

Stack:- (legacy class introduced in 1.0 version)

1) It is a child class of vector

2) Introduce in 1.0 v legacy class

3) It is designed for LIFO(last in fist order)

Ex :-

```
import java.util.*;
class Test
{
public static void main(String[] args)
{
Stack s=new Stack();
s.push("A");
s.push(10);
s.push("aaa");
System.out.println(s);
s.pop();
```

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

System.out.println(s.search("A"));

}

}
```

Cursors:-

The main purpose of the constructors is to retrieve the data from the collection objects.

There are three types of cursors present in the java language.

1. Enumaration

2. Iterator

3. ListIteator

Enumeration :-

1. It is used for only legacy classes(Vector,Stack)

2. Based on above reason the enumeration cursor is not a universal cursor

3. By using this cursor it is possible to read the data only it not possible to update the data an not possible to delete the data.

4. By using elements method we are getting enumaration object.

Ex:-

```
import java.util.*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

for (int i=0;i<10 ;i++ )

{

v.addElement(i);

}
```

```
System.out.println(v);
Enumeration e=v.elements();
while (e.hasMoreElements())
{
Integer i=(Integer)e.nextElement();
if (i%2==0)
{
System.out.println(i);
}
}
System.out.println(v);
}
}
```

Iterator:-

1. it is universal cursor we can apply any type of collection class.
2. By using this it is possible to read the data and remove the data.
3. We can use iterator() method to get the iterator object.

Ex:-

```
import java.util.*;
class Test
{
public static void main(String[] args)
{
Vector v=new Vector();
for (int i=0;i<10 ;i++ )
{
v.addElement(i);
}
```

```
System.out.println(v);
Iterator itr=v.iterator();
while (itr.hasNext())
{
Integer i=(Integer)itr.next();
if (i%2==0)
{
System.out.println(i);
}
else
itr.remove();
}
System.out.println(v);
}
}
```

LIstIterator :-

1. It is applicable for only list type of objects.
2. By using this it is possible to read the data upate the data and delete data also.
3. By using listIterator() method we are getting LIstIterator object

Ex :-

```
import java.util.*;
class Test
{
public static void main(String[] args)
{
Vector v=new Vector();
for (int i=0;i<10 ;i++ )
{
```

```
v.addElement(i);
}
System.out.println(v);
ListIterator litr=v.listIterator();
while (litr.hasNext())
{
Integer i=(Integer)litr.next();
if (i==0)
{
litr.add("veeru");
}
if (i==5)
{
litr.set("sambha");
}
if (i==9)
{
litr.remove();
}
}
System.out.println(v);
}
}
```

HashSet :-

Introduced in 1.2 v

Duplicate objects are not allowed if we are trying to insert duplicate values then we won’t get any compilation errors an won’t get any Execution errors simply add method return false.

Null insertion is possible

Heterogeneous objects are allowed

The under laying data structure is hashTable.

Insertion order is not preserved.

Ex :-

```
import java.util.*;

class Test

{

public static void main(String[] args)

{

HashSet h=new HashSet();

h.add("a");

h.add("a");

h.add("aaaa");

h.add(10);

h.add(null);

System.out.println(h);

}

}
```

LinkedHashSet :-

Introduced in 1.4 v

Duplicate objects are not allowed if we are trying to insert duplicate values then we wont get any compilation errors an won’t get any Execution errors simply add method return false.

Null insertion is possible

Heterogeneous objects are allowed

The under laying data structure is linkedList & hashTable.

Insertion order is preserved.

It is a child class of HashSet.

Ex :-

```
import java.util.*;
class Test
{
public static void main(String[] args)
{
LinkedHashSet h=new LinkedHashSet();
h.add("a");
h.add("a");
h.add("aaaa");
h.add(10);
h.add(null);
System.out.println(h);
}
}
```

TreeSet :-

1. The underlying data Structure is BalencedTree.
2. Insertion order is not preserved it is based some sorting order.
3. Hetrogeneous data is not allowed.
4. Duplicate objects are not allowed
5. Null insertion is possible only once.

Ex :-

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

```
TreeSet t=new TreeSet();

t.add(50);

t.add(20);

t.add(40);

t.add(10);

t.add(30);

System.out.println(t);

SortedSet s1=t.headSet(50);

System.out.println(s1); //[10,20,30,40]

SortedSet s2=t.tailSet(30);

System.out.println(s2); //[30,40,50]

SortedSet s3=t.subSet(20,50);

System.out.println(s3); //[20,30,40]

}

}
```

Map:-

1. Map is a child interface of collection.

2. Up to know we are working with single object and single value where as in the map collections we are working with two objects and two elements.

3. The main purpose of the collection is to compare the key value pairs and to perform necessary operation.

4. The key and value pairs we can call it as map Entry.

5. Both keys and values are objects only.

6. In entire collection keys can’t be duplicated but values can be duplicate.

HashMap :-

1. It used to hold key value pairs

2. Underlying data Structure is HashTable.

3. Duplicate keys are not allowed but values can be duplicated.

4. Insertion order is not preserved.

5. Null is allowed for key (only once)and allows for values any number of times.

6. Every method is non-synchronized so multiple Threads are operate at a time hence permanence is high.

Eg :-

```
import java.util.*;

class Test

{

public static void main(String[] args)

{

HashMap h=new HashMap();

h.put("sambha",100);

h.put("veeru",100);

h.put("durga",100);

System.out.println(h);

Set s=h.keySet();

System.out.println(s);

Collection c=h.values();

System.out.println(c);

Set s1=h.entrySet();

System.out.println(s1);

Iterator itr=s1.iterator();

while (itr.hasNext())

{

Map.Entry m1=(Map.Entry)itr.next();

System.out.println(m1.getKey()+"------"+m1.getValue());

if (m1.getKey().equals("sambha"))

{
```

```
m1.setValue("gayan TeamLead");

}

}

System.out.println(s1);

}

}
```

HashTable :-

1. It is a legacy class introduced in the 1.0 version.

2. Every method is synchronized hence only one thread is allow to access.

3. The performance of the application is low.

4. Null insertion is not possible if we are trying to insert null values we are getting NullPointerException.

Ex :-

```
import java.util.*;

class Test

{

public static void main(String[] args)

{

Hashtable h=new Hashtable();

h.put("hyd",100);

h.put("bang",200);

h.put("pune",300);

System.out.println(h);

System.out.println(h.contains(300));//true

System.out.println(h.containsValue(500));//false

Collection c=h.values();

System.out.println(c);

Set c1=h.keySet();
```

```
System.out.println(c1);
}
}
```

LinkedHashMap :-

1. It used to hold key value pairs
2. Underlying data Structure is HashTable & LinkedList.
3. Duplicate keys are not allowed but values can be duplicated.
4. Insertion order is preserved.

Eg :-

```
import java.util.*;
class Test
{
public static void main(String[] args)
{
LinkedHashMap h=new LinkedHashMap();
h.put("sambha",100);
h.put("veeru",100);
h.put("durga",100);
System.out.println(h);
Set s=h.keySet();
System.out.println(s);
Collection c=h.values();
System.out.println(c);
Set s1=h.entrySet();
System.out.println(s1);
Iterator itr=s1.iterator();
while (itr.hasNext())
{
```

```
Map.Entry m1=(Map.Entry)itr.next();

System.out.println(m1.getKey()+"------"+m1.getValue());

if (m1.getKey().equals("sambha"))

{

m1.setValue("Simon TeamLead");

}

}

System.out.println(s1);

}

}

==================================**********==========================
```