JAVA:

=====

What is java?

Object oriented Programming language and Platform independent language,

java was developed by james gosling in 1995.

jdk--1.7 java is object oriented programming language.

What is latest version of version?

jdk 1.8 --object oriented and Functional programming language

(Lambda expressions, streams, jondi API).

jdk -1.9 -- Object oriented and Functional Programming language.

--Additional functionality is called JS shell

To check which version of java is installed in system:

---------------------------------------------------------

cmd-> javac -version --jdk -- to compile

cmd-> java -version --jre -- to run

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)

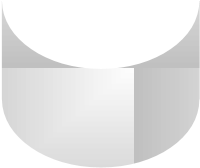
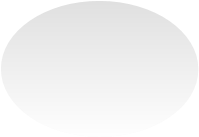
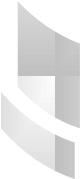
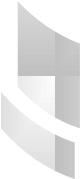
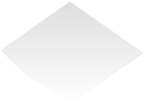
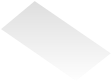
**J2SE:-**

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.
    3. For the standalone applications doesn’t need internet connections.
    4. It is a local application it doesn’t need any other external application support.
    5. This type of the applications we can launch by using the command line or by using the executable jar.



Client Database

**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.
    3. The application which is present in the internet is called the web application.
    4. Web application can be launched by using HTTP driven. HTTP request sent to the Servlet present in the server side.

**Web-application architecture:-**

Client

server

database

**request**

**Hibernate**

**response**

**Jdbc**

Html

Css

Jsp

velocity

Used to

store the

data.

(

)

java

.net

)

(

(

php

)

**Client:-**

The person who is sending the request is called client. All web browsers come under the clients. Ex: - InternetExploral, MozillaFrefox, opera………..etc

**Server:-**

The server contains the applications. The main purpose of the server is

* 1. It will contain the application.
  2. Take the request from the client.
  3. Based on the taken request it will identify the project resource and execute that project resource.
  4. By executing the project some response will be generated that response is dispatched to the client browser.

Ex: - Tomcat, GlassFish, WebLogic, JBOSS, WebSphere………..etc

**Database:-**

Database is used to store the details like client details, application details, registration details……etc.

Ex: - Oracle, MySQL………..etc

**J2ME:-**

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

**JAVA Features:-**

1. Simple
2. Object Oriented
3. Platform Independent
4. Architectural Neutral
5. Portable
6. Robust
7. Secure
8. Dynamic
9. Distributed
10. Multithread
11. Interpretive
12. High Performance
13. **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.

1. **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.

class data

class Test

{ Test t=new Test();

class data; t

}

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

1. **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.

JVM

windows

JVM

linux

JVM

JVM

macos

solaris

**source file**

**compilation**

**JVM understandable file**

**output console**

Test.java

Test.class

Javac

exe

exe

exe

exe

1. **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.

1. **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.

1. **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

* 1. JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.
  2. 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.

1. **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.

1. **Dynamic:-**

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

1. **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.

1. **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.

1. **Interpretive:-**

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

1. **High Performance:-**

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

##### Steps to Design a First Application:-

|  |  |
| --- | --- |
| Step 1 :- | Select an Editor. |
| Step2:- | Write a Program & save the program with .java extension |
| Step3:- | Compile the program. |
| Step4:- | Execute the program. |

**Step1:- Select an Editor**

Editor is a tool or software it will provide very good environment to type the java application

Ex: - Editors-----Notepad, Notepad++, Edit Plus---etc

Editors--- Eclipse, My eclipse, Net Beans----etc

**Step 2:- Write the program and save the program**

Write the java program based on the java API (Application Programming Interface) rule and regulations.

Java is a case Sensitive Language while writing the program we have to take care about the case (Alphabet symbols)

After writing the program we have to save the program while saving the program we have to consider following steps

Check whether the source file contain the public class or not

1. If the source file contain public class the name of the public class and the name of the Source file must be same (publicClassName.java). Otherwise we are getting compilation error.
2. if the source file does not contain any public class at that situation we can save the source file with (anyName.java)

**Step3:- Compile the program**

To single source file

Ex: - Javac filename.java

Javac Test.java

The java compiler goes to Test.java file and it will search for syntactical error if the syntactical errors are presented the java compiler raise compilation error if there is no syntactical errors are presented at that situation the java compiler converts the .java files into the .class file.

Note: - the .class file generation totally based on the number of classes present in the source it is not depending the number of classes present in the source file it is not depending on the name of the file. To compile multiple source files at a time Javac \*.java

Ex.java

Javac Ex.java

Class A

{

}

Class B

{

}

Class C

{

}

compilation

A.class

B.class

C.class

**Step4:- Execute the program**

Java class-name

Java Test

Whenever we are typing the Test class in the command prompt first it will search for the Test class if the Test class is available then it will search for the Main method if the main method is there the program execution starts from main method onwards.

The JVM will search for the .class file if the .class file is not available then JVM will raise an exception

**Exception in thread “main” java.lang.NoClassDefFoundError**

The JVM will search for the .class file if the .class is available the JVM will search for the main method if the main method is not available then the JVM will raise an Exception. **Exception in thread “main” java.lang.NoSuchMethodError: Main**

**Class Contains Five elements:-**

Class Test

{

1. variables
2. methods
3. constructors
4. instance blocks
5. static blocks

}

**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 (“welcome to java”);

System.out.print (“core java”);

Output: - welcome to javacorejava

**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 (“welcome to java”);

System.out.println (“core java”);

Output: - welcome to java

Corejava

**Identifiers:-**

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

class Test Test-------identifier

{

void add() add-------identifier

{

int a=10; a-------identifiers int b=20; b------identifiers

}

};

**Rules to declare identifiers:-**

1. The java identifiers should not start with numbers, it may start with alphabet symbol and underscore symbol and dollar symbol.

1. Int abc=10;-----valid
2. Int 2abc=20;----not valid
3. Int \_abc=30;----valid
4. Int $abc=40;----valid
5. Int @abc=50;---not valid

2. The identifier will not contains symbols like +, - , . , @ , # , \*………………….

1. The identifier should not duplicated.

class Test

{

void add()

{

int a=10; int a=20; the identifier should not be duplicated.

}

};

1. In the java applications it is possible to declare all the predefined class names and predefined interfaces names as an identifier. But it is not recommended to use.

class Test

{

public static void main(String[] args)

{

int String=10; //predefind String class int Serializable=20; //predified Seriaiable class float Exception=10.2f; //predefined Exceptionclass

System.out.println(String);

System.out.println(Serializable);

System.out.println(Exception);

}

};

JAVA NAMING CONVENSIONS:-

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 inner word also start with capital letters.

Ex: String

StringBuffer

FileInputStream

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

Ex: - post() toString()

toUpperCase()

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

Ex: - pageContent bodyContent

empSalary

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

Ex: MIN\_PRIORITY

MAX\_PRIORITY

NORM\_PRIORITY

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

Ex: java.awt Java.io , com.org.apache

NOTE:-

The coding standards are applicable for predefined library not for user defined library .But it is recommended to follow the coding standards for user defined library also.

**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

\*/

1. **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

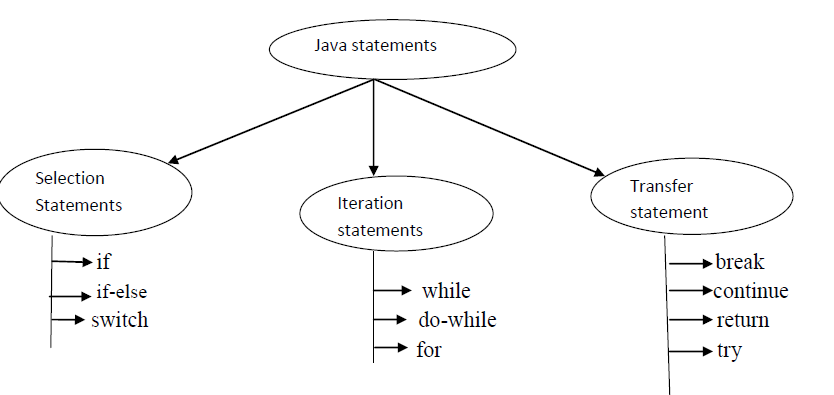
Syntax: - /\*\*…………….line-1

\*……………..line-2

\*………………line-3

\*/

## **Java Statements:-**



**If syntax:-**

if (condition)

{

if body;

}

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

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

**Ex 1:-** class Test

{

public static void main(String[] args)

{

Inta=10; if (a>5)

{

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

}

}

}

**Ex 2:-** class Test

{

public static void main(String[] args)

{

if (true)

{

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

}

System.out.println("hi how are you");

}

}

**Ex 3:-**

class Test{

public static void main(String[] args)

{

if (false)

{

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

}

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

}

}

**If-else syntax:-**

if (condition)

{

if body;(true body)

}

else

{

else body;(false body)

}

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

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

**Ex:-**

class Test {

public static void main(String[] args)

{

inta=10; intb=20; if (a<b)

{

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

}

else

{

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

}

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

}

}

**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
   1. Byte
   2. Short
   3. Int
   4. Char
   5. String(allowed in 1.7 version)
5. If the case is matched particular case will be executed if there is no case is matched default case is executed.

**Syntax:-**

switch (argument)

{

Case label1 : sop(“ “);

break;

Case label2 : sop(“ “);

break;

default : sop(“ “);

}

**Eg:**

class Test {

public static void main(String[] args)

{

int a=50; switch (a)

{

case 10:System.out.println ("10");

break;

case 20: System.out.println("20");

break;

case 30: System.out.println("30");

break;

case 40: System.out.println("40");

break;

default: System.out.println("default");

}

}

};

**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.

**Flow of execution in for loop:**

**-**

for (initialization ;condition ;increment/decrement )

{

Body;

}

1

2

5

4

7

3

6

Step1:- initialization is down

Step 2:- condition will be checked.

Step 3:- if the condition is true body will be executed. If the condition is false loop stopped.

Step 4:- after body increment/decrement part will be executed.

Step 5:- condition will be checked.

Step 6:- if the condition is true body is executed.

Step 7:- after body increment/decrement part will be executed.

The above process is repeated until the condition is false. If the condition is false the loop is stopped

**Ex 1:-**

**normal for loop**

class Test

{

public static void main(String[] args)

{

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

{

System.out.println("Hi");

}

}

}

**While:-**

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

**Syntax:- //initialization**

while (condition)

{

body;

increment/decrement

}

**Ex 1 :-**

class Test {

public static void main(String[] args)

{

int i=0;

while (i<10)

{

System.out.println("hi");

i++;

}

}

}

**Ex 2:-represent infinite loop**

class Test

{

public static void main(String[] args)

{

int i=0; while (true)

{

System.out.println("hi");

i++;

}

}

}

**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("Hi");

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**

**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.

1. Inside the switch statement.
2. 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);

}

}

}

**Key words:-**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Keywords  for  Modifier | Keywords  for  Flow  Control | Keywords for  Exception  Handling | Keywords for  Class | Keywords  for  Object | Keywords  for primitive data Types |
| Public private protected  static abstract final native volatile  synchronized  transient strictfp (11) | If  else  switch break case default for do while continue  (10) | try  catch finally  throw throws    (5) | Import  class interface extends package implements  (6) | New  instanceof super  this      (4) | Byte  short  int  long  float  double char boolean  (8) |
| Unused keywords | Metho level | Reserved literals | 1.5 version keywords |
| goto const (2) | Void return  (2) | true false null (3) | assert  enum  (2) |

**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 | Default values |
| byte short int long float double char  boolean | 1  2  4  8  4  8  2  NA | -128 to 127  -32768 to 32767  -2147483648 to 2147483647  -2`31 to 2`31-1  -3.4e38 to 3.4e308  -1.7e308 to 1.7e308  0 to 65535  Not Applicable | 0  0  0  0  0.0  0.0  Single space character  false |

**Syntax: - datatype name-of-variable=value/literal;**

Ex:-

int a=10;

Int--------Data Type

a----------variable name =----------assignment

10--------constant value

; ----------statement terminator

**Literals:-**

Literal is a constant value assigned to the variables.

‘a’----------------char literal

10---------------integral literal

false-----------boolean literal

10.2345--------double literal

**Floating point literal & double literal:-**

By default the decimal point values represent double so if we want to assign the floating point values to the variables we must attach the suffix **F or f** to the number.

If we are not providing the number we will get compilation error possible loss of precision.

double d=100.9898-------------good

float f=10.897--------------------compilation error(possible loss of precision)

float f=12345.67890f;---------good suffix with f it is optionally to attach the d or D with double double value

double d=1234.5678;-------------good

double d=1234567.7654321d;-----good

**Boolean literal:-**

The Boolean values are true or false.

boolean b=true; ------good

**Char literal:-**

A character literal is represented by a character in single quote.

char ch=’a’;

char ch1=’#’; char ch=’1’;

class Test

{

public static void main(String[] args)

{

char ch1='"';

System.out.println((int)ch1); char ch2='\'';

System.out.println((int)ch2);

}

}

**String literal:-**

A string literal represented by group of character in double quotes. Default value for the String is null.

String str=”welcome”;

class Test

{

public static void main(String[] args)

{

String str1="hello";

System.out.println(str1+"soft");

System.out.println("welcome"+"java home");

}

}

**Types of variables:-**

* Variables are used to store the values. By storing that values we are achieving the functionality of the project.
* While declaring variable we must specify the type of the variable by using data type’s concept.

In the java language we are having three types of variables

1. Local variables
2. Instance variables
3. Static variables

**Local variables:-**

1. The variables which are declare inside a method & inside a block & inside a constructor is called local variables
2. The scope of local variables are inside a method or inside a constructor or inside a block.
3. We are able to use the local variable only inside the method or inside the constructor or inside the block only.
4. Local variable must be initialized before use it.

Ex:-

class Test{

public static void main(String[] args)

{

int a=10; Local variables int b=20;

System.out.println(a+b);

}

}

**Instance variables:-**

1. The variables which are declare inside a class and outside of the methods is called instance variables.
2. We are able to access instance variables only inside the class any number of methods.

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 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 – it not recommended

**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.

### **Class vs Object:-**

* Class is a group of objects that have common property.
* Java is classes based language we are able to design the program by using classes and objects.
* Object is a real world entity. Object orientation is methodology to design a program by using classes and objects.
* Object is physical entity whereas 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 characteristics**

1. State(represent data of an object)
2. Behavior(represent behavior of an object)

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

PEN (object):-

State: - name raynolds, color red etc……

Behavior: - used to write

example:

=======

Chair is an Object

properites of chair:- color,material,shape,wheels..etc

Actions of chair -- movieable (), seatAdjustable ()

class Chair

{

String color="black";

String material="plastic";

int wheels = 4; //properties

public void movieable(){ //actions

System.out.println("Chair is moveable");

}

}

How to create object for class:

===============================

1) new operator:

= ===============

Using new operator we can create the Object for a class

Syntax:

=======

classname referencename = new classname();

ex:

===

Chair c = new Chair ();

To call the instance data members and instance methods;

========================================================

Syntax:

=======

referencename.variablename //to call instance variables

referencename.methodname () // to call instance methods

ex:

====

System.out.println(c.color);

System.out.println(c.wheels); //to call instance variables

c.movieable();//to call instance method

To call static data members and static methods:

==============================================

Syntax:

========

classname.variable --- to call a static variable

classname.methodname --to call static methods.

Typecasting:

=========

There are two types of typecasting

1) Widening casting or implicit casting:

================================

converting one datatype of a variable into other data type.

Note:

====

It convert lower datatype to higher datatype

byte->short->int->long->float->double

or

char->int->long->float->double

2) Narrowing casting or explicitly casting:

================================

Converting higher datatype of a variable into Lower datatype.

With help of type cast () operator .

byte->short->int->long->float->double --in reverse

char->int->long->float->double

syntax:

======

(specify the datatype)

Example :

========

1) write a java program to convert one datatype into another data type.

sol:

===

public class ExampleOnCasting{

public static void main(String args[]){

int a=10;

double b = a; //widening casting or implicit casting

System.out.println("the value of a variable is "+b);

double d = 20.0;

int c = (int)d; //narrowing casting or explicit casting

System.out.println("the value of a variable is "+c);

float x =(float) d; //converting double to float

System.out.println(x);

long aa = 30;

float y= aa; //converting long to float

System.out.println(y);

}

}

**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)

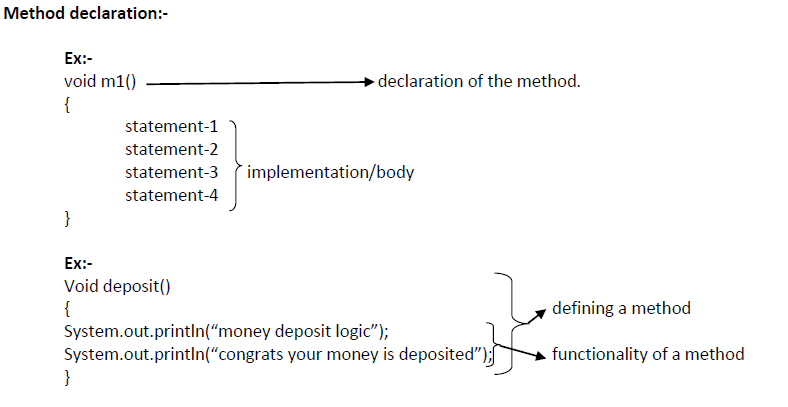
**Method Signature:-**

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

Ex:-

m2(int,int)------Method Signature

m1();------------------Method signature



There are two types of methods:-

**Instance method**

**Ex: void m1 () {**

**}**

**Static method**

Ex: static void m1 () {

}

**Oops concepts:-**

**1) Class**

**2) Object**

**3) Inheritance**

**4) Polymorphism**

**5) Abstraction**

**6) Encapsulation**

**Inheritance:-**

The process of getting properties and behaviors from one class to another class is called inheritance.

Properties: variables

Behaviors: methods

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

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

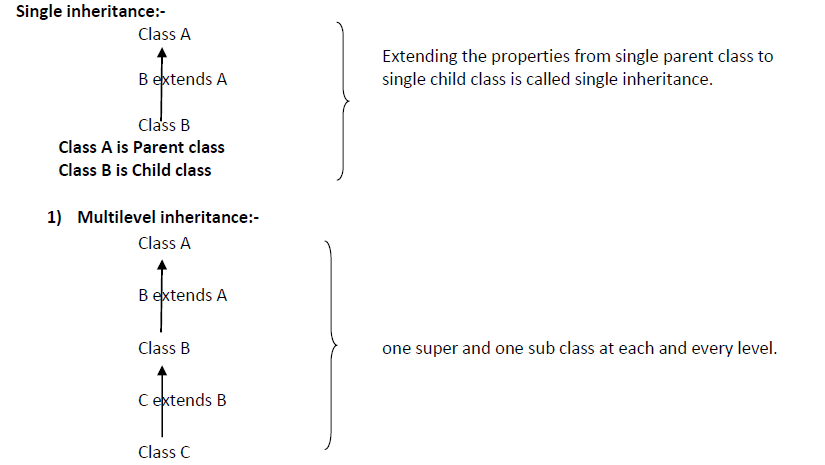
3. Inheritance is also known as **is-a** relationship means two classes are belongs to the same hierarchy.

4. By using **extends** keyword we are achieving inheritance concept.

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

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

**Types of inheritance:-**



**2) 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 so one class can extends only one class at time it is not possible to extends more than one class at time.

class B extends A-------possible

class C extends B------possible

class C extends A,B--not possible

class A class B

C extends A,B

class C

**3) Hierarchical inheritance:-**

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

class A

B extends A

C extends A

D extends A

class B class C class D

**4)** **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.

Class A

B extends A

Class B

C extends B

Class C

D extends C

E extends C

Class E

class D

}

;

multilevel inheitance

hierarchical inheritance

**Note:-**

1. Every class in the java programming is a child class of object.

2. The root class for all java classes is Object class.

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

4.It is recommended to create the object for child class(sub-class)

**POLYMORPHISM:-**

1. One thing can exhibits more than one form called polymorphism.
2. The ability to appear in more forms.
3. Polymorphism is a Greek word poly means **many** and morphism means forms.

Polymorphism

Polymorphism

Compile Time Polymorphism Run Time Polymorphism

(Early binding) (Late binding)

(Static binding) (Dynamic binding)

Ex:-Method Overloading Ex:-Method Overriding

**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
     1. Provide the different number of arguments to the same methods.
     2. 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.

Automatic promotion types in method overloading:

==========================================

IF calling any method in method overloading ,compiler may not found the method based on referenced type compiler will not throw any error message immediately, it will check for next level reference datatype based on the any method available it will execute otherwise will get error.

**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.

The parent class method is called-------------overridden method

The child class method is called---------------overriding method

For A super class Reference variable holding its sub class Object Execution:

===========================================================

SuperClassName referencevariable= new SuperClassName ();

referencevariable.methodName ();

1) First it will check the method reference in Super class and based on

Method signature if available it will check same methodSignature in child class,

If available it will execute child class method.

2) If method signature not available in super class and the same method signature available in child class, it will not execute child class method it will throw an error.

**Abstraction:-**

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

Ex:-

a. Bank ATM Screens (Hiding the 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

Interface:

==========

--> An interface which contains set of abstract methods.

-->Interface used for SRS (software requirement specification or abstract).

-->An interface contain variables and these variable must be static and final

Syntax:

=======

public interface InterfaceName{

//variables;

//abstract methods;

}

-->once the interface is created who will provide the implementation,

Developer is responsible to provide implementations for the interface

--> How to provide implementation

To provide implementations A class must **implements** the

Interface and provide the implementation for all the abstract method.

Note:

=====

1) If we not provide implementation for any one abstract method then declare the class as abstract.

2) We cannot create the object for abstract class directly.

3) To create the object for the abstract class, create one more class which extends from abstract class and provide the implementation for the abstract method

4) Create an object for new created class which extends from abstract class

5) Call the data members and member methods of its own class as well as abstract class

Example:

========

1) create a interface named as MyApp

2) this interface contain two abstracts methods

a)public abstract int getSum(int x,int y);

b)public abstract int getAddition(int x,int y,int z);

c) public void m1();

3) create a class named as ExampleOnInterface which implements

MyApp interface

note:- provide implementation for all abstract methods.

4) create an Object for implementation class and call those methods.

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

**1) Normal classes**

**2) Abstract classes**

**Normal classes:-**

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

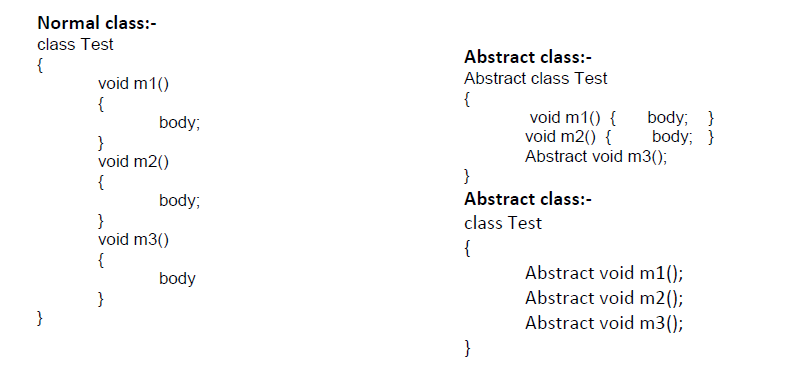
**Abstract class:-**

Abstract class is a java 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 unimplemented 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.



**Ex:- the abstract class contains abstract methods for that abstract methods provide the implementation in child classes.**

abstract class Test

{

abstract void m1();

abstract void m2();

abstract void m3();

}

class AbstractDemo extends Test

{

void m1()

{

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

}

void m2()

System.out.println("m2-method");

}

void m3()

{

System.out.println("m3-method");

}

public static void main(String[] args)

{

AbstractDemo ad=new AbstractDemo();

ad.m1();

ad.m2();

ad.m3();

}

};

**Ex :- 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.**

abstract class Test

{

abstract void m1();

abstract void m2();

abstract void m3();

}

abstract class AbstractDemo1 extends Test

{

void m1()

{

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

}

void m2()

{

System.out.println("m2-method");

}

};

class AbstractDemo extends AbstractDemo1

{

void m3()

{

System.out.println("m3-method");

}

public static void main(String[] args)

{

AbstractDemo ad=new AbstractDemo();

ad.m1();

ad.m2();

ad.m3();

}

};

**Note:-**

**In the above program**

**abstract Test---------------------------****not possible to create an object.**

**abstract AbstractDemo1-----------****not possible to create an object.**

**AbstractDemo------------------------****possible to create an object.**

**Note:-**

**We can take the any number of child classes but we have to provide the implementation for each and every abstract method.**

**Encapsulation:-**

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

We are able to provide more encapsulation 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. (javabean class)—(In hibernate (Pojo class))

Ex:-

class EncapsulationExample

{

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("hi");

int num=e.getSid();

String name=e.getSname();

System.out.println(num);

System.out.println(name);

}

};

**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.

4) Constructor is a special type method whose name is same as class name.

5) Constructor does not return any value not even void (empty) also.

6) A constructor is used to initialize the object

Note:

=====

Initialize the object means, allocating memory for instance variables (heap) and the instance variables will store with it default values based on it types.

There are two types of constructors

1) Default Constructor.

2) Parameterized constructor

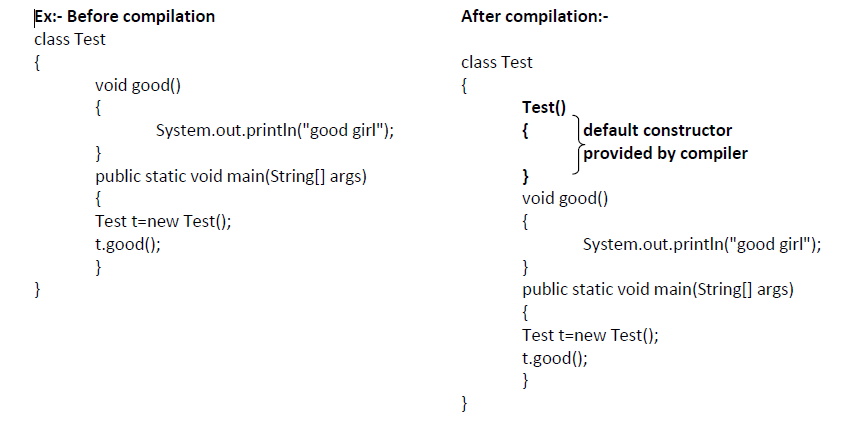
**Default Constructor:-**

1) In the java programming if we are not providing 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.



**Parameterized constructor:**

A constructor is defined with some arguments with in class.

Ex:

Parameterized constructor is executed.

class Test

{

Test(int i)

{

System.out.println(i);

}

void good()

{

System.out.println("good girl");

}

public static void main(String[] args)

{

Test t=new Test(10);

t.good();

}

}

**this keyword**:

=============

1) this keyword which represents current class object or instance

2) using this keyword we can call instance methods and instance variable

3) using this() method we can call constructors.

Note:- this keyword should be a first line of statement in constructor

Example:

--------

public class ExampleOnThis{

int a=10;

String s = "hello";

public ExampleOnThis(){

System.out.println("we are in default constructor");

System.out.println(a+ " "+s);

}

public ExampleOnThis(int x,String s){

System.out.println("we are in parameterized constructor");

}

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

ExampleOnThis eot = new ExampleOnThis();

}

super keyword:

==============

->super keyword is used to refer parent class object.

->using super we can call super class instance variables and

super class instance methods

->using super () method we can call super class constructors.

because in inheritance sub class cannot override the constructors.

to call a super class method:

======================

super.methodname(); --we need to call in any instance methods of sub class

to call super class instance variable:

=============================

super.variableName --we need to call in any instance variable of sub class

to callsuper class constructor:

========================

super() ---it will call default constructor of super class by

default without specifying.

super(any arguments) -- it will call parameterized constructor of super class

based on arguments type

note:

====

1)it will call default constructor of super class by default without specifying

in subclass default constructor.

2)super() or super(any arguments ) --should be first line of statement

in sub-class constructors.

Example:

========

public class ExampleOnSuper

{

int a=10;

public ExampleOnsuper(){

System.out.println("we are in default constructor of ExampleOnsuper");

}

public void display(){

System.out.println("we are in display method of ExampleOnsuper");

}

}

public class Test3 extends ExampleOnSuper{

int a=20;

public Test3(){

System.out.println("we are in default constructor of Test class");

}

public void display(){

System.out.println("we are in display method of test class");

}

public static void main(String args[]){

Test t = new Test();

}

}

**Final keyword:**

**--------------------**

* Final is the modifier applicable for classes, methods and variables (for all instance, Static and local variables).
* If a class is declared as final, then we cannot inherit that class i.e., we cannot create any child class for that final class.
* Every method present inside a final class is always final by default but every variable present inside the final class need not be final.
* The main advantage of final modifier is we can achieve security as no one can be allowed to change our implementation.
* But the main disadvantage of final keyword is we are missing key benefits of Oops like inheritance and polymorphism. Hence is there is no specific requirement never recommended to use final modifier.

**Packages**

Information regarding 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 shareability of the project will be increased.

6) package statement should be the first line in java class.

Syntax:-

package package\_name;

Ex: - package com.training;

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 packages contains 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 (interface)

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:-**

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

Example:

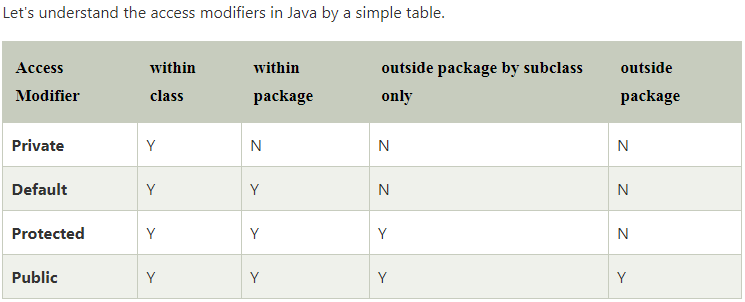
package com.training.dao

# **Access Modifiers**

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.



**Arrays:-**

1) Array is a final class inheritance is not possible.

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

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

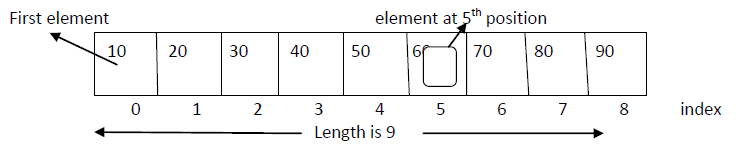
4) The items presented in the array are classed elements. Those elements can be accessed by index values. The index is begins from (0).

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]; //declaring size

a[0]=10;

a[1]=20;

a[2]=30;

a[4]=40;

**Declaration of two dimensional array:-**

int[][] a;

int [][]a;

int a[][];

int []a[];

**Java.lang.String**

String:-

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

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

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

4) In String class .equals () is used for content comparison.

Constructors of string class:-

public String(java.lang.String);

public String(char[]);

public String(char[] ,int ,int );

**Creation of String object:-**

To create a object for string class we are having two approaches

1) without using new operator(by using literal)

2) by using new operator

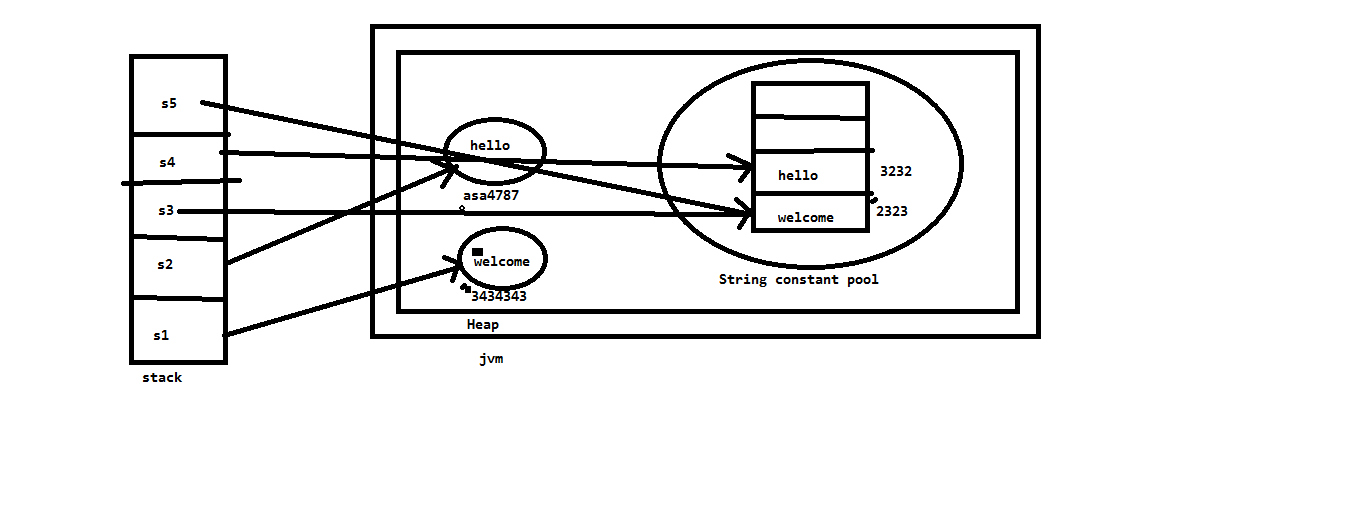
**Creating a string object without using new operator (by using literal):-**

String str1=”hello”;

String str2=”welcome”;

Note:

Whenever we create string literal first jvm goes to SCP (String constant pool) and check if the string is already present in the pool or not. If it is available it returns the existing reference from the pool, if it is not available a new String object is created



public boolean equals():-

a) String class equals () method is used for content comparison. It returns true of false value after comparison.

b) The commonly used method to perform comparison.

c) We are comparing total content called deep comparison.

d) At the time of comparison the fallowing possibilities are occur.

True----------two Strings are same

False---------two strings are not equals

== Operator:-

It is used for reference comparison. Hence we can call shallow comparison.

Methods:

------------

**concat():-** concat() method present in the String class and it is used to combine the two String.

**length():-** It is used to find out the length of the string

**charAt(int):-**

By using above method we are able to extract the character from particular index position.

**split(String):-**

By using split() method we are dividing string into number of tokens.

**public Boolean equalsIgnoreCase():-**

By using above method we are comparing the strings

**public bytes[] getBytes():-**By using this method we are converting String into the byte[] .the main aim of the converting String into the byte[] format is some of the networks are supporting to transfer the data in the form of bytes only at that situation is conversion is mandatory.

**trim():-**

1) trim() is used to remove the trail and leading spaces

2) this method always used for memory saver

**replace(char oldchar,char newchar)**

**replace(String oldString,String newString):-**

by using above method we are replacing the particular character of the String.

And particular portion of the string.

**toUpperCase() and toLowerCase():-**

The above methods are used to convert the lower case to the uppercase and uppercase to lowercase character.

**Java.lang.String.endsWith() and Java.lang.String.startsWith():-**

**endsWith()**

is used to find out if the string is ending with particular character/string or not.

**startsWith()** used to find out the particular String starting with particular character/string or not.

**substring(int startingposition) &**

**substring(int startingposition,int endingposition)**

By using above method we are getting substring from the whole String.

In the above methods

starting position parameter value is including

ending position parameter value is excluding

**StringBuffer:-**

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

2) StringBuffer is a final class so it can’t be inherited.

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

4) StringBuffer .equals () method is used for reference comparison.

**Constructors:-**

1) StringBuffer sb=new StringBuffer();

2) StringBuffer sb1=new StringBuffer(int capacity);

3) StringBuffer sb2=new StringBuffer(String str);

**StringBuffer is mutable:-**

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

Note 1:- it is possible to change the content of StringBuffer Object

Note2:- StringBuffer .equals () is used for reference comparison (address comparison)

Note 3:- == operator is used for reference comparison (address comparison)

Methods:

------------

reverse():-By using this method we can display the string in reverse.

append():-

By using this method we can append the any values at the end of the string.

Insert():-

By using above method we are able to insert the string any location of the existing string.

**public** **class** ExampleOnStringBuffer {

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

StringBuffer sb = **new** StringBuffer("hello");

System.***out***.println(sb);

sb.append(" welcome to java");

System.***out***.println(sb);

StringBuffer sb1 = **new** StringBuffer("hello");

StringBuffer sb2= **new** StringBuffer("welcome");

StringBuffer sb3 = **new** StringBuffer("hello");

System.***out***.println(sb1==sb3);

System.***out***.println(sb1.equals(sb3));

System.***out***.println("reverse the string : "+sb.reverse());

System.***out***.println("insert : "+sb1.insert(2,"hi"));

}

}

**Wrapper classes**

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

2) All wrapper classes present in the java.lang package.

3) Int,byte…. Acts as a primitives we can make the primitives into the objects is called wrapper class the wrapper classes are Integer, Short-----.

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

5) Data types and corresponding wrapper classes:-

byte ----- Byte

short ----- Short

int ----- Integer

long ----- Long

float ----- Float

double ----- Double

boolean ----- Boolean

char ------ Character

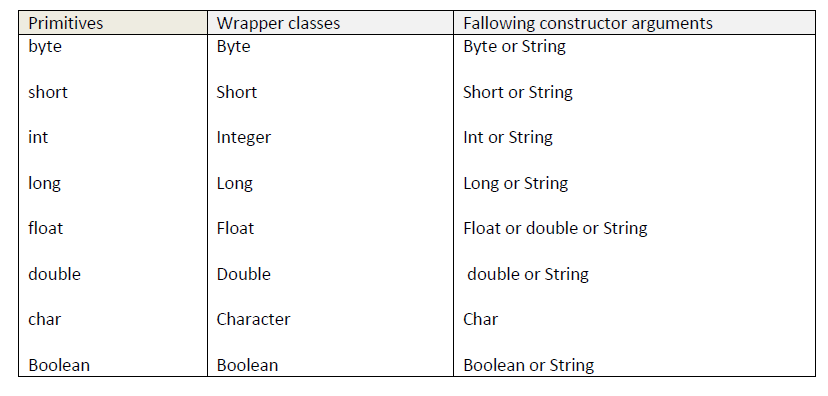
6) Byte,Short,Integer,Long,Float,Double these are child classes of Number class.

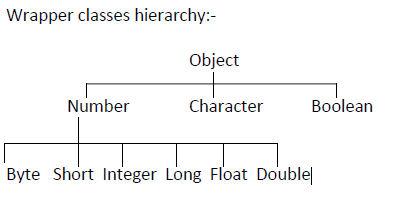
**Constructors of wrapper classes :-**

All most all wrapper classes contain two constructors:-

1. Integer i=new Integer(10);

2. Integer i=new Integer(“10”);





**public** **class** ExampleOnWrapperClasses {

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

**int** i=10;

String s="20";

String s1 ="20.0";

//String s="sda";

//Converting Primitive Datatype into Object

Integer iobj = **new** Integer(i);

System.***out***.println(iobj);

//converting StringObject into an IntegerObject.

Integer iobj1 = **new** Integer(s);

System.***out***.println(iobj1);

Float f1 = **new** Float(s1);

System.***out***.println(f1);

**int** i1=Integer.*parseInt*(s);

System.***out***.println("i1 is "+i1);

**float** f2 = Float.*parseFloat*(s1);

System.***out***.println("f2 is : "+f2);

}

}

**Java.util.Scanner:-**

By using Scanner class we are able to divide the String into the number of tokens.

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

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

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

**import** java.util.Scanner;

**public** **class** ExampleOnScan1 {

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

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter the employeeId ");

**int** empId = sc.nextInt();

System.***out***.println("enter the employeeName");

String empName=sc.next()+sc.nextLine();

System.***out***.println("Enter the employee Salary");

**float** empSal=sc.nextFloat();

System.***out***.println("employee Details are : ");

System.***out***.println("employee Id is : "+empId);

System.***out***.println("employee Name is : "+empName);

System.***out***.println("employee Salary is : "+empSal);

sc.close();

}

}

**Exception Handling**

Information regarding Exception:-

1) Dictionary meaning of the exception is abnormal termination.

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

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

4) Exception is nothing but an object.

5) Exception is a class present in java.lang package.

6) All the exceptions are nothing but objects called classes.

7) Whenever user is entered invalid data then Exception is occur.

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

9) 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

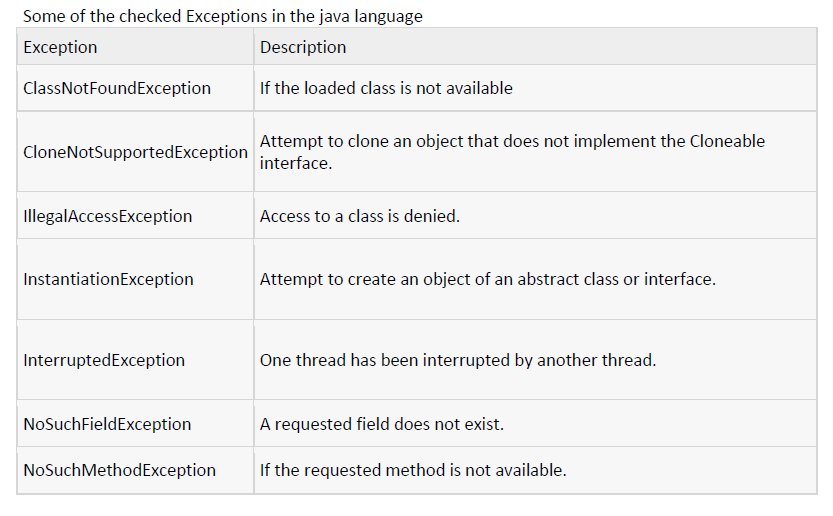
2) Unchecked Exception

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 Checked Exceptions.

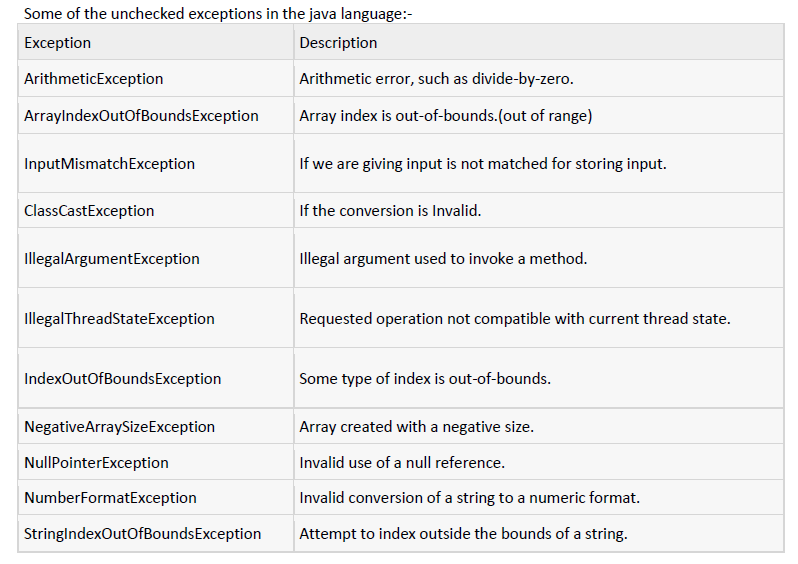
Ex: - IOException, SQLException etc……….



**Unchecked Exception:-**

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

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



**Difference between Exception and Error:-**

**Exception:-**

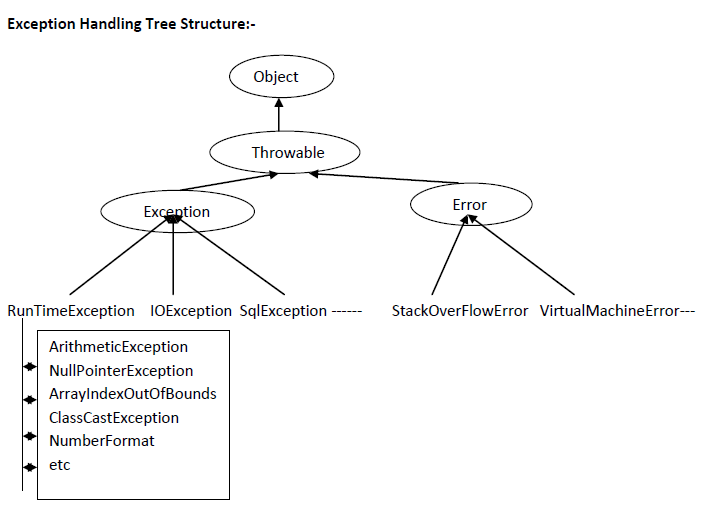
An exception is unwanted unexpected event these are caused by the programmer mistake. Exceptions are recoverable.

Ex: - IOException, SQLExcetion, RuntimeExecption etc……………

**Error:-**

Errors are caused by lack of system resources. These are non-recoverable.

Ex: - StackOverFlowError, AssertionError etc…………



**Note:-**

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

2) Root class for all Exception hierarchy is Throwable 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 aim 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 (error code) of the program and catch block contains handling code of the program.

2) Catch block code is an 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

{

Code to run [break;]

}

catch(ExceptionName reference\_variable)

{

Code to run if an exception is raised

}

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

class Test

{

public static void main(String[] args)

{

System.out.println(“Xyz”);

System.out.println("Software");

System.out.println(10/0);

System.out.println("solutions");

}

}

**Output:-**

Xyz

Software

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

Note: - if we are not using try-catch it is always abnormal termination if an Exception raised.

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

Handled by JVM type of the Exception description

**After try catch:-**

1) If we are taking try-catch the program goes to normal termination. Because the risky code (error 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("Xyz");

System.out.println("software");

try

{

System.out.println(10/0); //error code

}

catch (ArithmeticException e)

{

System.out.println("you are getting AE "+e);

}

System.out.println("solutions");

}

}

**Output:-**

Xyz

Software

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

Solutions.

**Ex 1:-**

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");

}

}

**Ex 2:-**

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. If the catch block is not matched the program is terminated abnormally.

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(ArithmeticException ae)

{

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

}

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

}

}

**Ex 3:-** if there is no exception in try block the catch blocks won’t be executed.

class Test

{

public static void main(String[] args)

{

System.out.println("program starts");

try

{

System.out.println("HI sir");

System.out.println("how r u");

}

catch(ArithmeticException ae)

{

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

}

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

}

}

Note:

In java class if we are declaring multiple catch blocks at that situation the catch block order should be child to parent shouldn’t be parent to the child.

**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;

}

**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 an Exception object implicitly. Whereas throw keyword creates an Exception object explicitly.

**Throws:-**

1) Throw keyword is used to create exception object explicitly. But the main purpose of the throws keyword is by passing 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.

**User defined Exceptions:-**

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

Ex: InvalidAgeException………..etc

**To create** user defined Exceptions:-

1) To create user defined exception we have to take a user defined class that is a sub class to the RuntimeExecption (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

for the creation of UncheckedException:-

**Default constructor approach**

class InvaliadAgeException extends RuntimeException

{

}

**Parameterized constructor approach**

class XXXException extends RuntimeException

{

XXXException(String str)

{

Super(str);

}

}

Note:-

for these type of user defined Exceptions no need of handling the Exception . Hence try-catch [or] throws keyword is not required.

Ex:-

public class InvaliedAgeException extends Exception

{

InvaliedAgeException(String str)

{

super(str);

}

//program that uses user defined Exception class

import java.util.\*;

class Test

{

static void validate(int age) throws InvaliedAgeException

{

if (age<18)

{

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

}

else

{

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

}

}

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

{

Scanner s=new Scanner(System.in);

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

int age=s.nextInt();

validate(age);

}

}

Or:

**import** java.util.Scanner;

**import** com.training.exception.InvalidAgeException;

**public** **class** ExampleOnUserDefinedException {

**public** **static** **void** main(String[] args) **throws** InvalidAgeException {

Scanner sc= **new** Scanner(System.***in***);

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

**int** age=sc.nextInt();

**if**(age<18) {

**throw** **new** InvalidAgeException("Age should be greater than 18 to vote");

}**else** {

System.***out***.println("Valid Age");

}

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

sc.close();

}

}

Exception Examples:

-----------------------------

**public** **class** ExampleOnExep1 {

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

System.***out***.println("statement -1");

**try** {

System.***out***.println(10/0);//error code

}**catch**(ArithmeticException e) {

System.***err***.println("denominator cannot be zero");

//System.out.println(e);

//e.printStackTrace();

System.***out***.println(e.getMessage());

}

System.***out***.println("rest of the code or statement-2");

}

}

2)

**public** **class** ExampleOnExcept2 {

**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**(ArithmeticException ae)

{

//System.err.println("we are getting exception");

ae.printStackTrace();

}**catch** (ArrayIndexOutOfBoundsException ae) {

System.***err***.println("we are getting exception from ArrayIndexOutOfBounds....");

}

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

}

}

3)

**public** **class** ExampleOnExcept3 {

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

System.***out***.println("statement-1");

**try** {

System.***out***.println(10/5);

**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**(ArithmeticException ae) {

// **TODO**: handle exception

ae.printStackTrace();

}**catch**(Exception e) {

e.printStackTrace();

}

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

}

}

4)

**public** **class** ExampleOnExcept4 {

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

System.***out***.println("statement");

**try** {

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

System.***out***.println(10/0);

}**catch**(ArithmeticException ae) {

ae.printStackTrace();

}

**finally** {

System.***out***.println("In finally block we will write clean code ");

}

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

}

}

5)

import java.io.IOException;

import java.sql.SQLException;

public class ExampleOnExcept5 {

public void methodOne(){

System.out.println("we are in methodOne");

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("statement-2");

}

public static void main(String[] args){

ExampleOnExcept5 e5 =new ExampleOnExcept5();

e5.methodOne();

}

}

6)

**public** **class** ExampleOnExcept6 {

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

**int** a=0;

**int** b=10;

**if**(a==0) {

**throw** **new** ArithmeticException("a cannot be Zero");

}**else** {

System.***out***.println("division of two numbers is "+b/a);

}

System.***out***.println ("rest of the line");

}

}

**Multi-Threading**

**Multi-Tasking:**

--------------

Executing several tasks simultaneously is the concept of multi-tasking.

There are two types of multi-tasking

1) Process-based multi-tasking

2) Thread Based multitasking.

Process Based Multitasking:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Executing several tasks simultaneously where each task is a separate independent program (process) is called process based multitasking.

ex:

While typing a program in the editor we can listen audio songs from the same system at the same time we can download a file from net, all these tasks will be executed simultaneously and independent of each other, hence it is process-based multi-tasking.

Process Based Multitasking is best suitable at OS (operating system) level.

Thread Based Multitasking:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Executing several task simultaneously where each task is a separate independent part of the same program is called Thread based multitasking. And each independent part is called a 'thread'.

Thread Based Multitasking best suitable at programmatic level.

Imp. Note:

------------

Whether it is process based or thread based the main objective of multi-tasking is to reduce response time of the system. And to improve performance.

The main important application area of multithreading are:

1) To develop multimedia graphics...

2) TO develop animations

3) TO develop video games.

4) TO develop webservers/application servers...etc

Note:

-----

90 percent in Api-- java inbuilt support for multithreading...

10 percent for programmer work

When compared with old languages developing, multi-threaded application in java is very easy, because java provides inbuilt support for multithreading with rich API ([Thread, Runnable, Thread Group...])

**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");

System.out.println ("hello");

}

}

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 following 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());

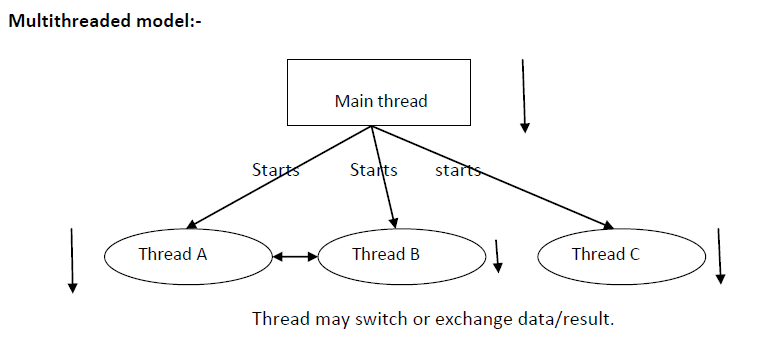
System.out.println("hi");

System.out.println("hello");

}

}

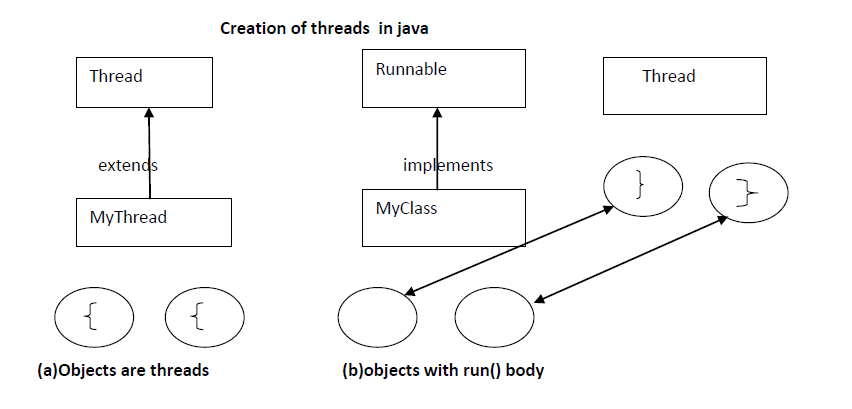
In the above program only one thread is available name of that thread is main thread.



**There are two different ways to create a thread is available**

1) Create class that extending standard 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 that is extending Thread class .**

Ex:-

class MyThread extends Thread

{

public void run()

{

System.out.println("business logic of the thread ");

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.

**Second approach to create thread implementing Runnable interface:-**

class MyClass implements Runnable

{

public void run()

{

System.out.println("business logic of the thread ");

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.

**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

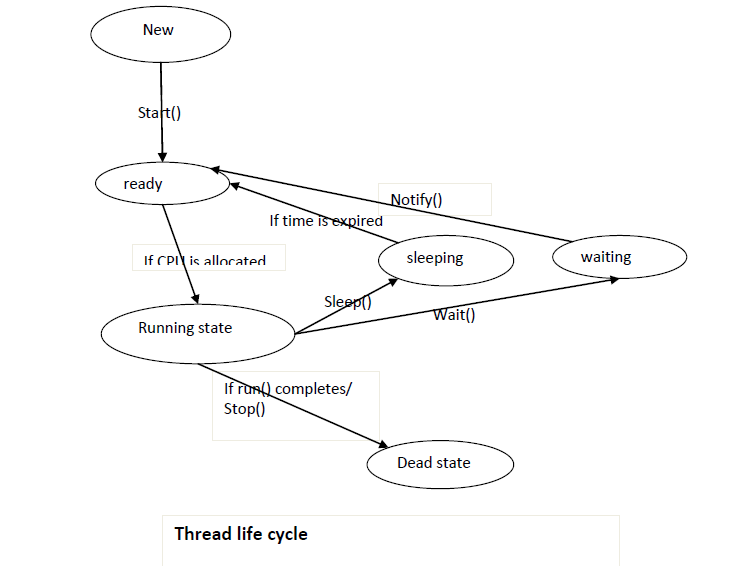
**Blocked State:-**

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.

**Thread life cycle:-**



**Thread Scheduler:-**

* Thread scheduler is a part of the JVM. It decides which thread is executed first and which thread is executed next.
* Only one thread is executed at a time.
* We can’t expect exact behavior of the thread scheduler it is JVM vendor dependent. So we can’t expect output of the multithreaded examples we can say the possible outputs.
* Thread Scheduler mainly uses preemptive (or) time slicing to schedule the threads.

**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.

**Java.lang.Thread.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");

}

}

}

**Java.lang.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 InterruptedException

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("hi");

Thread.sleep(3\*1000);}

catch(InterruptedException iee)

{

System.out.println("getting interrupted exception");

}

}

}

}

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.

Non-synchronized methods

void m1()

{ non-synchronized method any number of threads can access

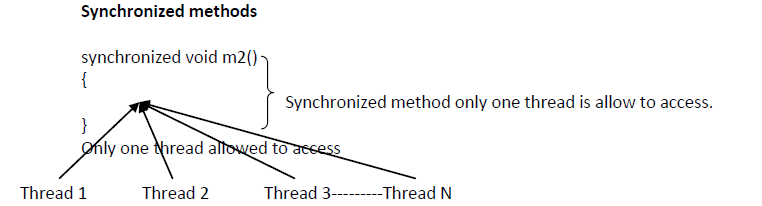
}

Every thread accessing simultaneously

Thread 1 Thread 2 Thread 3---------Thread N

1) In the above case multiple threads are accessing the same methods hence we are getting data inconsistency problems. These methods are not thread safe methods.

2) But in this case multiple threads are executing so the performance of the application will be increased.



1) In the above case only one thread is allow to operate on particular method so the data inconsistency problems will be reduced.

2) Only one thread is allowed to access so the performance of the application will be reduced.

3) If we are using above approach there is no multithreading concept.

Hence it is not recommended to use the synchronized modifier in the multithreading programming.

**Inter-thread communication in Java** is a technique through which multiple threads communicate with each other.

It provides an efficient way through which more than one thread communicate with each other by reducing CPU idle time. CPU idle time is a process in which CPU cycles are not wasted.

When more than one threads are executing simultaneously, sometimes they need to communicate with each other by exchanging information with each other. A thread exchanges information before or after it changes its state.

There are several situations where communication between threads is important.

For example, suppose that there are two threads A and B. Thread B uses data produced by Thread A and performs its task.

If Thread B waits for Thread A to produce data, it will waste many CPU cycles. But if threads A and B communicate with each other when they have completed their tasks, they do not have to wait and check each other’s status every time.

Thus, CPU cycles will not waste. This type of information exchanging between threads is called **inter-thread communication in Java**.

Two threads communicate each other by using wait(),notify(),notifyAll(),these methods available inside java.lang.Object class.

The thread which is excepting updating, it has to call wait() method and the thread which is performing updation it has to call notify() method. After getting notification the waiting thread will get those updations.

java.lang.Object

=================

public void wait();

public void wait(long mill sec);

public native void wait(long millsec,long nanosec);

public native void notify(); -- it call only one waiting thread

public void notifyAll(); -- it call multiple threads in waiting

based on priority

note:

====

when we are working with inter thread communication the methods should be synchronized otherwise you will get RE: IllegalMonitorStateException

Example:

========

1) Create a class Customer it contain two methods withdrawal () and deposit () and create a communication between two thread one thread has to call withdrawal () and another thread has to deposit the amount.

sol:

----

public class Customer{

int amount=10000;

public synchronized void withrawal(int amount){

if(this.amount<amount){

System.out.println("Customer has to wait untill

deposit the amount");

try{

wait();

Thread.sleep(1000);

}catch(Exception e){ system.out.println(e.getMessage())};

}

this.amount=this.amount-amount;

System.out.println("Balance amount is "+this.amount);

}

public synchronized void deposit(int amount){

this.amount= this.amount+amount;

System.out.println("Amount is depostied customer has to

withdrawal");

notify();

}

}

2) public class ExampleOnWithdrawal extends Thread{

Customer c;

public ExampleOnWithdrawal(Customer c){

this.c=c;

}

public void run(){

c.withdrawal(15000);

}

}

3) public class ExampleOnDeposit extends Thread{

Customer c;

public ExampleOnDeposit(Customer c){

this.c=c;

}

public void run(){

c.deposit(30000);

}

}

4) public class Test1{

public static void main(String args[]){

Customer c = new Customer();

ExampleOnWithdrawal t = new ExampleOnWithdrawal(c);

t.start();

ExampleOnDeposit t1 = new ExampleOnDeposit(c);

t1.start();

}

}

}

**Collections (java.util)**

**Limitations of array:-**

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

2) Arrays can hold homogeneous data only

3) Once we created an array no chance of increasing or 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

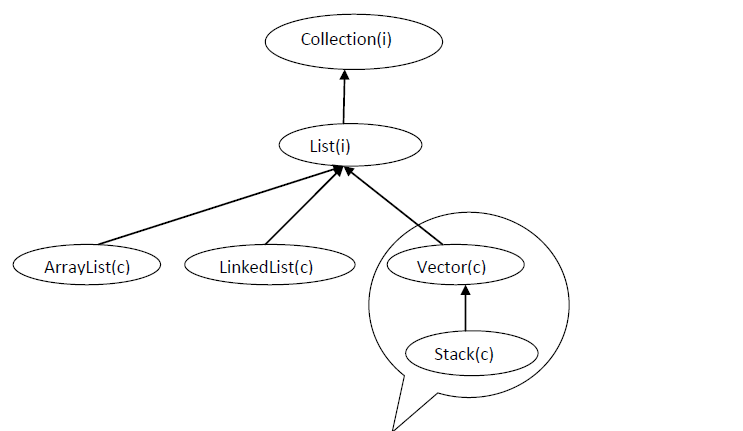
5. NavigableSet

6. Queue

7. Map

8. SortedMap

9. NavigableMap



**I --------------****Interface**

**c--------------****class**

**ArrayList:-**

class ArrayList extends AbstractList 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.

Examples:

------------

Ex1:

package com.training.list;

import java.util.ArrayList;

import java.util.Iterator;

public class ExampleOnArrayList1 {

public static void main(String[] args) {

ArrayList al = new ArrayList();

//store the elements into ArrayList we need to use add()

al.add(10);

al.add(30.0f);

al.add(40.0);

al.add("hi");

al.add('a');//

al.add(false);

al.add(10); //duplicate

al.add(null); //null value

System.out.println("size of any arraylist is : "+al.size());

System.out.println(al);

al.remove(4);

System.out.println("after delete the elements are: "+al);

al.set(6,"welcome");

System.out.println("after replaceing the element at position 6 is ");

System.out.println(al);

boolean b = al.contains(20);

System.out.println("element is searched : "+b);

Object o=al.get(3);

System.out.println(o);

System.out.println( "displying the elements from arraylist using iterator");

Iterator iobj =al.iterator();

while(iobj.hasNext()) {

System.out.println(iobj.next());

}

}

}

Ex2:

-----

package com.training.list;

import java.util.ArrayList;

import java.util.Iterator;

public class ExampleOnArrayList2 {

public static void main(String[] args) {

ArrayList<String> al = new ArrayList<String>();

al.add("hello");

al.add("smith");

al.add("laxmi");

al.add("martin");

al.add("john");

al.add("smith");

//al.add(10);

System.out.println("elements from arraylist is : "+al);

Iterator<String> iobj =al.iterator();

System.out.println("displaying the elements from Arraylist");

while(iobj.hasNext()) {

System.out.println(iobj.next());

}

System.out.println("elements from the arraylist using foreach loop");

for(String l:al) {

System.out.println(l);

}

}

}

Ex3:

-----

**package** com.training.list;

**public** **class** Employee {

**private** **int** empId;

**private** String empName;

**private** **float** empSal;

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Employee(**int** empId, String empName, **float** empSal) {

**super**();

**this**.empId = empId;

**this**.empName = empName;

**this**.empSal = empSal;

}

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** **float** getEmpSal() {

**return** empSal;

}

**public** **void** setEmpSal(**float** empSal) {

**this**.empSal = empSal;

}

}

**package** com.training.list;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**public** **class** ExampleOnArrayList3 {

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

ArrayList<Employee> al = **new** ArrayList<Employee>();

Employee e1= **new** Employee(101,"smith",3000);

Employee e2= **new** Employee(102,"john",1000);

Employee e3= **new** Employee(103,"martin",4000);

al.add(e1);

al.add(e2);

al.add(e3);

al.add(**new** Employee(104,"blake",5000));

System.***out***.println("size of an array is : "+al.size());

System.***out***.println(al);

System.***out***.println("Displaying the elements from the arraylist");

Iterator<Employee> iobj=al.iterator();

System.***out***.println("EMployee Details are : ");

**while**(iobj.hasNext()) {

Employee eobj =iobj.next();

System.***out***.println(eobj.getEmpId()+" "+eobj.getEmpName()+" "+eobj.getEmpSal());

}

}

}

**LinkedList:-**

Class LinkedList extends AbstractSequentialList implements List, Deque, Queue

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 order is preserved.

Ex1:

package com.training.list;

import java.util.Iterator;

import java.util.LinkedList;

public class ExampleOnLinkedList1 {

public static void main(String[] args) {

LinkedList ll = new LinkedList();

ll.add("smith");

ll.add(10);

ll.add(20.0f);

ll.add('c');

ll.add(true);

ll.add(30.0);

System.out.println("no of elements are : "+ll.size());

System.out.println(ll);

ll.addFirst("ramu");

ll.addLast("bye");

System.out.println(ll);

System.out.println("search is available ?"+ll.contains('d'));

ll.remove(4);

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

ll.set(4,"hi");

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

System.out.println("displaying the elements using the iterator");

Iterator iobj =ll.iterator();

while(iobj.hasNext()) {

System.out.println(iobj.next());

}

}

}

Ex2:

**package** com.training.list;

**import** java.util.LinkedList;

**public** **class** ExampleOnLinkedList2 {

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

LinkedList<Integer> ll = **new** LinkedList<Integer>();

ll.add(10);

ll.add(**new** Integer(20));

ll.add(40);

System.***out***.println("no of elements are : "+ll.size());

ll.addFirst(30);

ll.addLast(50);

System.***out***.println(ll);

System.***out***.println("Displaying the elements using foreach loop");

**for**(Integer i:ll) {

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

}

}

}

**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.

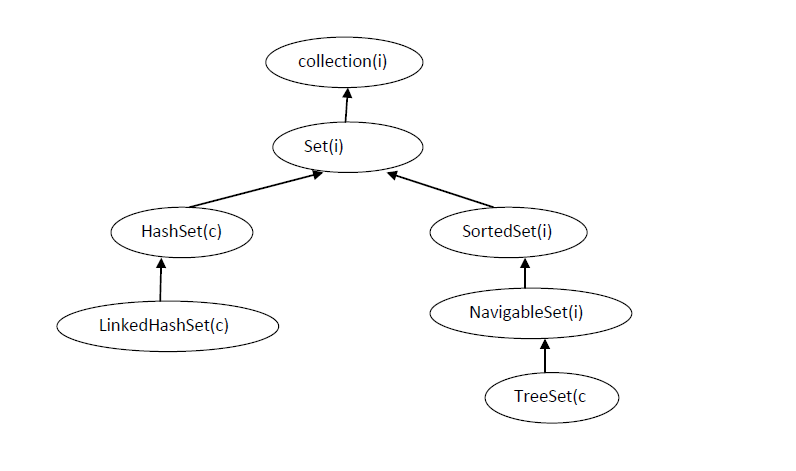
**ListIterator:-**

1. It is applicable for only list type of objects.

2. By using this it is possible to read the data update the data and delete data also.

3. By using listIterator () method we are getting LIstIterator object

**SET**



**HashSet:-**

1. Introduced in 1.2 v

2. 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.

3. Null insertion is possible

4. Heterogeneous objects are allowed

5. The under laying data structure is hashTable.

6. Insertion order is not preserved.

Ex1:

package com.training.set;

import java.util.HashSet;

import java.util.Iterator;

public class ExampleOnHashSet1 {

public static void main(String[] args) {

HashSet hs = new HashSet();

hs.add(10);

hs.add("smith");

hs.add(true);

hs.add('d');

hs.add(24.0f);

hs.add(30.0);

hs.add(10);

hs.add(null);

System.out.println("no of elements are : "+hs.size());

System.out.println(hs);

hs.remove('d');

System.out.println("after remove is :");

System.out.println(hs);

System.out.println("Search element is found ? "+hs.contains("smi"));

System.out.println("dislaying the elements using iterator");

Iterator iobj =hs.iterator();

while(iobj.hasNext()) {

System.out.println(iobj.next());

}

for(Object o:hs) {

System.out.println(o);

}

}

}

Ex2:

------

**package** com.training.set;

**import** java.util.HashSet;

**public** **class** ExampleOnHashSet2 {

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

HashSet<String> hs = **new** HashSet<String>();

hs.add("smith");

hs.add("martin");

hs.add("ramu");

hs.add(**new** String("hello"));

hs.add(**null**);

hs.add("smith");

**char** c[]= {'c','a','b','d'};

String s= **new** String(c);

hs.add(s);

hs.add("10");

System.***out***.println("no of elements are : "+hs.size());

System.***out***.println("displaying the elements using foreach loop");

**for**(String s1:hs) {

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

}

}

}

**LinkedHashSet:-**

1. Introduced in 1.4 v

2. 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.

3. Null insertion is possible

4. Heterogeneous objects are allowed

5. The under laying data structure is LinkedList & hashTable.

6. Insertion order is preserved.

7. It is a child class of HashSet.

Ex1:

package com.training.set;

import java.util.Iterator;

import java.util.LinkedHashSet;

public class ExampleOnLinkedHashSet1 {

public static void main(String[] args) {

LinkedHashSet lhs = new LinkedHashSet();

lhs.add("smith");

lhs.add(10);

lhs.add(true);

lhs.add(30.0f);

lhs.add(null);

lhs.add(10);

lhs.add('a');

System.out.println("no of elements are : "+lhs.size());

System.out.println(lhs);

lhs.remove(true);

System.out.println("after delete : "+lhs);

System.out.println("displaying the elements using iterator");

Iterator iobj =lhs.iterator();

while(iobj.hasNext()) {

System.out.println(iobj.next());

}

}

}

**TreeSet:-**

1. The underlying data Structure is BalancedTree.

2. Insertion order is not preserved it is based some sorting order(Ascending).

3. Heterogeneous data is not allowed.

4. Duplicate objects are not allowed

5. Null insertion is possible only once.

Ex1:

package com.training.set;

import java.util.Comparator;

import java.util.Iterator;

import java.util.TreeSet;

public class ExampleOnTreeSet1 {

public static void main(String[] args) {

TreeSet<Integer> ts = new TreeSet<Integer>();

ts.add(10);

ts.add(40);

ts.add(60);

ts.add(70);

ts.add(5);

ts.add(10);

ts.add(10);

System.out.println("No of elements are : "+ts.size());

ts.remove(60);

System.out.println("after remove : "+ts);

System.out.println("Displaying the elements using the iterator");

Iterator iobj =ts.iterator();

while(iobj.hasNext()) {

System.out.println(iobj.next());

}

System.out.println("Displaying the elements using descendingIterator");

Iterator<Integer> iobj1=ts.descendingIterator();

while(iobj1.hasNext()) {

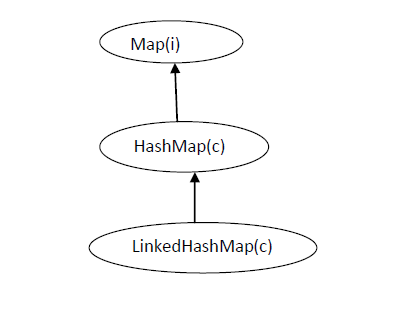
System.out.println(iobj1.next());

}

}

}

**Map interface:-**



**Map:-**

1. Map is a not 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 or entryset.

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

Ex1:

package com.training.map;

import java.util.Collection;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

public class ExampleOnHashMap1 {

public static void main(String[] args) {

HashMap hm = new HashMap();

hm.put("smith", 10);

hm.put(101,"hello");

hm.put(true,"hi");

hm.put(10,10);

hm.put(null,null);

hm.put(10,10.0f);

System.out.println("no of emlements are "+hm.size());

System.out.println(hm);

System.out.println("based on key get the value :"+hm.get("smith"));

hm.remove(true);

System.out.println("after remove is : ");

System.out.println(hm);

//get the only keys

Set s= hm.keySet();

System.out.println("keys are : "+s);

//get the only values;

Collection c=hm.values();

System.out.println("values are :"+c);

//get the entire entry set

Set s1= hm.entrySet();

System.out.println("displaying the elements using the iterator");

Iterator iobj=s1.iterator();

while(iobj.hasNext()) {

Map.Entry me =(Map.Entry)iobj.next();

System.out.println(me.getKey()+" "+me.getValue());

}

}

}

**LinkedHashMap:-**

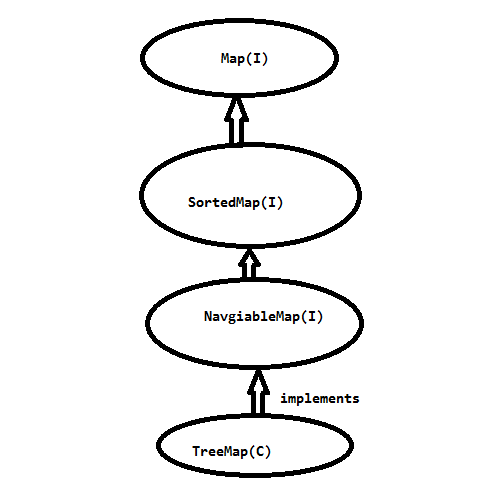
1. It used to hold key value pairs

2. Underlying data Structure is HashTable & LinkedList.

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

4. Insertion order is preserved.

**TreeMap:-**



**TreeMap:-**

1) The underlying data structure is RED-BLACK tree.

2) Duplicate keys are not allowed but values can be duplicated.

3) Insertion order is not preserved and all entries will be inserted according to some sorting order of keys.

4) Keys are homogeneous.